

Composants électroniques programmables sur PC



**CONSTRUISEZ VOS PROGRAMMEURS :
EPROM, PAL, PIC, EEPROM SÉRIE, MAC, ETC.**



Composants électroniques programmables sur PC

Mémoires EPROM ou EEPROM, PAL, GAL, EPLD, et bien sûr microcontrôleurs : autant de familles de composants électroniques programmables à l'intérieur desquels il est possible d'intervenir profondément grâce au programmeur approprié.

Avec l'aide de votre compatible PC, cette nouvelle édition de *Composants électroniques programmables*, vous apprend à « écrire dans le silicium » pour concevoir et produire vous-même des circuits intégrés parfaitement introuvables dans le commerce.

Outre une présentation détaillée des principales familles de composants programmables, vous y trouverez tous les plans des programmeurs nécessaires, y compris les tracés des circuits imprimés.

La disquette rassemble tous les programmes chargés de leur pilotage, et même quelques « grands » logiciels professionnels : compilateur logique, éditeur de fichiers, etc.

Il n'en faut pas davantage pour vous doter à très bon compte de votre propre « fabrique » de circuits intégrés personnalisés !

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES



Patrick GUEULLE

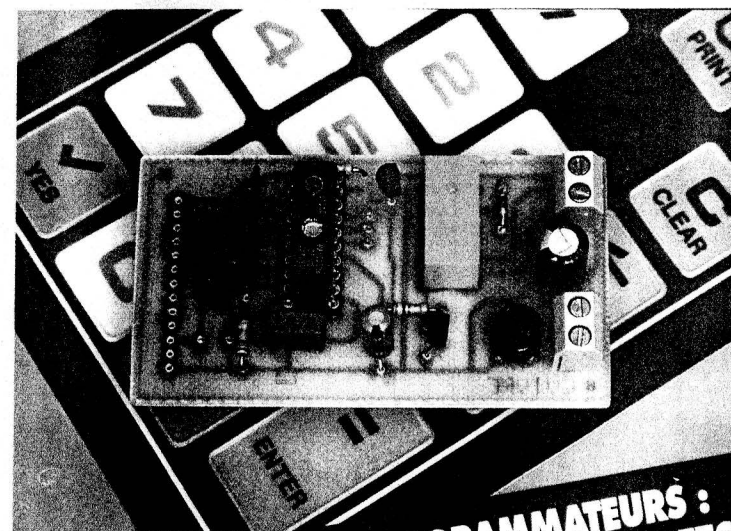
Patrick GUEULLE

COMPOSANTS ÉLECTRONIQUES PROGRAMMABLES SUR PC

Patrick GUEULLE

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES

Composants électroniques programmables sur PC



**CONSTRUISEZ VOS PROGRAMMEURS :
EPROM, PAL, PIC, EEPROM SÉRIE, MAC, ETC.**

CE LIVRE
CONTIENT
UNE DISQUETTE

Du même auteur dans la même collection

Cartes magnétiques et PC
Cartes à puces
Logiciels PC pour l'électronique
Montages à composants programmables
Télécommandes
PC et cartes à puce
Alimentations à piles et accu
CB antennes
Répondeurs téléphoniques

PATRICK GUEULLE

Ingénieur EFREI

COMPOSANTS ÉLECTRONIQUES PROGRAMMABLES SUR PC

ETSF

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES

Toutes les marques citées dans cet ouvrage sont des marques déposées par leurs propriétaires respectifs.

Nous vous rappelons, par ailleurs, les termes de l'article 47 de la loi du 3 juillet 1985 :

«Toute reproduction autre que l'établissement d'une copie de sauvegarde par l'utilisateur, ainsi que toute utilisation d'un logiciel non expressément autorisée par l'auteur ou ses ayants droit, est passible des sanctions prévues par la loi».

Ce pictogramme mérite une explication. Son objet est d'alerter le lecteur sur la menace que représente pour l'avenir de l'écrit, particulièrement dans le domaine de l'édition technique et universitaire, le développement massif du **photocopillage**.

Le Code de la propriété intellectuelle du 1er juillet 1992 interdit en effet expressément la photocopie à usage collectif sans autorisation des ayants droit. Or, cette pratique s'est généralisée dans les établisse-

ments d'enseignement supérieur, provoquant une baisse brutale des achats de livres et de revues, au point que la possibilité même pour les auteurs de créer des œuvres nouvelles et de les faire éditer correctement est aujourd'hui menacée.

Nous rappelons donc que toute reproduction, partielle ou totale, de la présente publication est interdite sans autorisation du Centre français d'exploitation du droit de copie (CFC, 3 rue Hautefeuille, 75006 Paris).



© DUNOD, Paris, 1997.

© E.T.S.F., Paris, 1991, pour la première édition.

ISBN 2-10-023987-2

Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite selon le Code de la propriété intellectuelle (Art L 122-4) et constitue une contrefaçon réprimée par le Code pénal. • Seules sont autorisées (Art L 122-5) les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective, ainsi que les analyses et courtes citations justifiées par le caractère critique, pédagogique ou d'information de l'œuvre à laquelle elles sont incorporées, sous réserve, toutefois, du respect des dispositions des articles L 122-10 à L 122-12 du même Code, relatives à la reproduction par reprographie.

TABLE DES MATIÈRES

Avant-propos

7

CHAPITRE

1	Introduction aux composants programmables	9
	L'évolution des circuits électroniques	10
	L'évolution des circuits intégrés	10
	Les circuits intégrés personnalisés	11
	Les circuits intégrés programmables	11
2	Les mémoires programmables	13
	Les familles de mémoires programmables	14
	Que contiennent les mémoires ?	15
	Les représentations usuelles	16
	La mise en fichiers	19
	Les applications des mémoires	22
	Les logiciels d'aide au développement	23
	L'EPROM, mémoire programmable par excellence	26
	Les programmeurs	38
	L'effacement des EPROM	42
	Les émulateurs d'EPROM et les RAM à pile	45
	Les mémoires EEPROM série	46
3	Les réseaux logiques programmables	57
	Notion de réseau logique programmable	58
	Les architectures de base	59
	Les PAL, réseaux logiques programmables par excellence	62
	Les PAL les plus populaires	65
	Les PAL universels et les GAL	71
	Que contiennent les PAL et les GAL ?	75
	Les logiciels d'aide au développement	75
	Les programmeurs	81
	Les « grands » EPLD	82
	Les pLSI et ispLSI de Lattice	84

AVANT-PROPOS

Fabriquer ses propres circuits intégrés spécifiques est un rêve que bien peu d'électroniciens peuvent se vanter de n'avoir jamais caressé.

Grâce aux *Composants Électroniques Programmables*, la réalité technique se rapproche de plus en plus de ce rêve, apportant des solutions fort élégantes à de multiples problèmes jusqu'alors imparfaitement résolus.

La mise en œuvre de ces composants fait cependant appel à des méthodes et à des outils qui n'ont plus rien de commun avec les techniques traditionnelles : il n'est évidemment pas question d'opérer au fer à souder à l'intérieur d'un circuit intégré ! Pour définir le composant personnalisé qui pourra, dans un second stade, être utilisé exactement comme un circuit intégré ordinaire, on aura dans un premier temps surtout recours au clavier d'ordinateur. Il faudra ensuite faire appel à un appareil très particulier nommé « programmeur » pour « écrire dans le silicium » le schéma ainsi développé.

Cet ouvrage a été conçu comme un outil d'initiation à la technique des composants programmables, avec le souci constant de permettre à ses lecteurs d'échapper au « droit d'entrée », parfois élevé, que certains fournisseurs s'efforcent d'instaurer.

Nous n'avons donc nullement la prétention de cataloguer ici tous les composants programmables existants ! Il en apparaît d'ailleurs constamment de nouveaux. Nous avons préféré sélectionner les plus courants, ceux-là mêmes que l'on peut se procurer assez facilement à des prix raisonnables, et ignorer ceux dont les procédures de programmation restent entourées d'un mystère excessif.

Dans tous les cas, l'indispensable programmeur pourra ainsi être construit à peu de frais, ou acheté sous la forme d'un *starter kit* très abordable.

Par la suite, ceux de nos lecteurs qui le souhaiteront seront parfaitement préparés pour partir à la découverte de composants encore plus élaborés, exigeant par contre le recours à des programmeurs « universels » considérablement plus coûteux.

4	Les microcontrôleurs	95
	Les systèmes à microprocesseur	96
	Les microcontrôleurs	98
	Les microcontrôleurs EPROM et OTP	99
	Les logiciels et les systèmes d'aide au développement	100
	Les programmeurs pour microcontrôleurs	105
	Les microcontrôleurs PIC	106
5	Réalisation de programmeurs	111
	Un programmeur d'EPROM	112
	Un lecteur d'EPROM	122
	La programmation et la lecture des RAM ZEROPOWER	131
	Une alimentation de sécurité pour programmation d'EPROM	134
	La programmation des ispGAL 22V10	138
	La programmation des ispLSI 1016 et 2032	145
	Un programmeur de PIC	146
	Un programmeur des EEPROM série	151
	Le logiciel de programmation	154
6	La disquette du livre	155

1 INTRODUCTION AUX COMPOSANTS PROGRAMMABLES

L'évolution des circuits électroniques	10
L'évolution des circuits intégrés	10
Les circuits intégrés personnalisés	11
Les circuits intégrés programmables	11

2	Les mémoires programmables	13
3	Les réseaux logiques programmables	57
4	Les microcontrôleurs	95
5	Réalisation de programmeurs	111
6	La disquette du livre	155

L'ÉVOLUTION DES CIRCUITS ÉLECTRONIQUES

L'électronique moderne est en grande partie une affaire de circuits intégrés : les composants discrets ont cédé beaucoup de terrain à leur profit, tout en demeurant indispensables dans un certain nombre de domaines bien précis. La plupart du temps, on ne peut, en fait, se contenter d'assembler purement et simplement des circuits intégrés standards. Il faut ajouter un certain nombre de composants discrets pour achever d'adapter le circuit à l'application envisagée.

La tendance la plus marquante de ces dernières années est l'intégration galopante des circuits. Grâce à des techniques qui se perfectionnent sans relâche, un seul circuit intégré peut actuellement contenir l'équivalent de tout un micro-ordinateur de moyenne puissance. C'est énorme !

Parallèlement, les techniques numériques s'imposent dans un nombre croissant de domaines, souvent au détriment des solutions analogiques qui résistent toutefois encore fort bien dans un certain nombre d'applications.

Enfin, le logiciel prend de plus en plus d'importance par rapport au matériel. Bien souvent, c'est dans un programme de microprocesseur que réside tout le secret d'un circuit, dont la partie matérielle est des plus banale. Or, il est infiniment plus facile de copier un programme que de relever le schéma d'un circuit imprimé un tant soit peu complexe. Cela pose un délicat problème de protection des secrets de fabrication.

L'ÉVOLUTION DES CIRCUITS INTÉGRÉS

Avec l'accroissement massif de leur densité d'intégration, les circuits intégrés deviennent de plus en plus spécialisés. Si une quadruple porte NAND à deux entrées peut servir à construire une quasi-infinité de montages, il n'en va, évidemment, pas de même d'un contrôleur d'écran-vidéo ou d'un synthétiseur de son !

Malgré l'immense variété des produits spécifiques disponibles sur le marché, le concepteur n'est jamais assuré de trouver le composant miracle qui résoudra à lui seul la totalité de son problème. Et quand bien même ce serait le cas, il faut compter avec les difficultés d'approvisionnement, les ruptures de stock, le manque de « secondes sources », et surtout les brusques abandons de fabrication.

Il est, en effet, extrêmement dangereux de faire reposer toute la faisabilité d'une réalisation sur un composant disponible auprès d'un unique fabricant. Outre le fait que l'absence de concurrence peut à tout moment faire flamber les prix, le risque est grand de voir la fabrication abandonnée en l'espace de quelques mois, sans aucun signe avant-coureur, ni recours d'aucune sorte.

On doit alors perdre tout espoir de poursuivre la fabrication au-delà des stocks disponibles, tandis que se posent pour l'avenir de sérieux problèmes de maintenance.

Faut-il pour autant se rabattre uniquement sur les composants « passe-partout » communs à toutes les marques ? Évidemment non, car leur manque de spécialisation alourdit considérablement les schémas tout en les rendant copiables par le premier venu.

LES CIRCUITS INTÉGRÉS PERSONNALISÉS

Lorsque des fabrications en grande série sont assurées sur une longue période, une solution efficace consiste à faire produire industriellement un circuit intégré sur mesure ou un ASIC (*Application Specific Integrated Circuit*). C'est extrêmement contraignant et assez risqué, mais très rentable lorsqu'il s'agit de centaines de milliers de pièces.

Les investissements de départ sont lourds, mais on peut définir un composant parfaitement adapté à l'usage prévu. Il ne sera pas disponible dans le commerce, et restera quasiment incopiable. Par contre, toute modification interne entraîne à nouveau une procédure longue et coûteuse. Le droit à l'erreur n'existe pas, et on ne peut guère envisager de perfectionner le produit d'une série à l'autre.

En tout état de cause, cette démarche est pratiquement inapplicable aux petites et moyennes séries, et à plus forte raison aux prototypes.

LES CIRCUITS INTÉGRÉS PROGRAMMABLES

Le concept de logique programmable est bien connu, et représente l'opposé de la logique câblée.

Dans un système logique câblé, le comportement du montage est entièrement fixé par l'interconnexion d'un certain nombre de composants. Les modifications sont malaisées et la copie une simple question de patience.

Un système programmé est constitué d'une unité centrale (microprocesseur) équipée de périphériques d'entrée-sortie et exécutant un programme enregistré dans une mémoire non volatile. C'est ce programme qui définit, presque à lui seul, le comportement du système. La simple réécriture de ce logiciel permet d'apporter des modifications profondes sans intervenir en aucune façon sur la partie matérielle, c'est-à-dire sur le circuit électronique. Une même carte peut tout aussi bien servir à gérer une chaudière qu'un composeur automatique de numéros de téléphone, tout dépend du contenu de sa mémoire !

Lorsque l'unité centrale et ses périphériques, plus évidemment de la mémoire, sont réunis dans un même boîtier, on parle volontiers de « microcontrôleur » ou de *monochip*, pouvant être programmé en vue d'une infinité d'applications. Des techniques ayant été développées pour rendre leurs programmes incopiables, la partie vitale de tels systèmes peut facilement être mise à l'abri des imitations abusives. C'est un très gros avantage.

Entre les circuits logiques câblés et les systèmes microprogrammés existe cependant une technique intermédiaire qui se taille un franc succès : les réseaux logiques programmables. Il ne s'agit plus de mémoires, mais de circuits intégrés dont le schéma interne peut être modifié de l'extérieur par des manœuvres simples, et pas forcément irréversibles.

Cela permet réellement de produire par soi-même, que ce soit à l'unité ou en série, des composants personnalisés sans avoir à supporter de frais fixes au départ comme pour les « sur mesure ». Très courante en milieu professionnel, cette technique se répand de plus en plus chez les amateurs avertis. Avec cet ouvrage, nous espérons contribuer à la démocratiser encore davantage.

1	Introduction aux composants programmables	9
---	---	---

2 LES MÉMOIRES PROGRAMMABLES

Les familles de mémoires programmables	14
Que contiennent les mémoires ?	15
Les représentations usuelles	16
La mise en fichiers	19
Les applications des mémoires	22
Les logiciels d'aide au développement	23
L'EPROM, mémoire programmable par excellence	26
Les programmeurs	38
L'effacement des EPROM	42
Les émulateurs d'EPROM et les RAM à pile	45
Les mémoires EEPROM série	46

3	Les réseaux logiques programmables	57
4	Les microcontrôleurs	95
5	Réalisation de programmeurs	111
6	La disquette du livre	155

Composants de plus en plus « stratégiques », les mémoires sont indispensables dans de nombreux domaines de l'électronique, d'ailleurs nullement limités à l'informatique.

L'amélioration des technologies de fabrication et l'explosion de la demande ont entraîné une baisse des prix, une augmentation des capacités, et des innovations qui rendent possibles des applications inimaginables il y a seulement quelques années.

Les mémoires pouvant être qualifiées de « programmables » occupent une place de choix dans cette famille de composants.

LES FAMILLES DE MÉMOIRES PROGRAMMABLES

Les différentes familles de mémoires sont couramment identifiées par des sigles d'origine anglaise, dont il est important de bien connaître la signification.

Distinguons tout d'abord les mémoires vives, dans lesquelles on peut indifféremment écrire ou lire mais qui perdent leur contenu dès la coupure de leur alimentation, des mémoires mortes dont le contenu est plus ou moins définitivement figé, et que l'on utilise essentiellement en lecture.

Les mémoires vives (MEV) sont volontiers appelées RAM (*Random Access Memories*) tandis que les mémoires mortes (MEM) sont souvent nommées ROM (*Read Only Memories*).

Les RAM elles-mêmes se classent en deux catégories : les RAM dynamiques (DRAM) et les RAM statiques (SRAM). Nettement plus simples à produire, les DRAM atteignent des capacités se chiffrant en mégabits, tout en coûtant beaucoup moins cher que les SRAM. En revanche, elles nécessitent un « rafraîchissement » permanent des données qu'elles contiennent, ce qui complique notablement leur mise en œuvre. En principe, les RAM ne sont donc pas des « composants programmables » au sens qui nous intéresse (composants pouvant être modifiés de façon durable par l'utilisateur lui-même), mais nous verrons qu'il existe des exceptions qui confirment la règle.

Du côté des ROM, il faut distinguer les mémoires mortes programmables une seule fois, des mémoires reprogrammables après effacement (REPROM).

Appartiennent à la première catégorie les ROM « masquées », dont le contenu est fixé dès la phase de fabrication en série, et

les PROM (*Programmable ROM*) « à fusibles » que l'utilisateur peut programmer lui-même, mais de façon irréversible.

Les plus connues des mémoires reprogrammables sont les EPROM (*Erasable PROM*), effaçables en bloc par exposition aux ultra-violets et reprogrammables des centaines de fois, mais qui existent désormais en version OTP (*One Time Programmable*). Dépourvues de fenêtre transparente et donc présentées en boîtier plastique économique, les EPROM OTP sont ineffaçables et donc programmables une seule fois.

Effaçables, en tout ou partie, de façon purement électrique et souvent même « transparente » pour l'utilisateur, les EEPROM (*Electrically Erasable Programmable ROM*) ou E2PROM offrent des alternatives intéressantes lorsque de fréquentes modifications sont nécessaires.

Les mémoires FLASH, pour leur part, peuvent être assimilées à des EPROM dont l'effacement, global et instantané, se fait électriquement. Leur franc succès dans certaines applications n'a toutefois pas « tué » les EPROM comme certains l'avaient prédit un peu hâtivement.

À mi-chemin entre les RAM et les ROM, les RAM CMOS à pile lithium incorporée méritent une mention à part. Pouvant être écrites comme des RAM puis traitées comme des ROM, elles sont généralement considérées comme plus performantes que les EEPROM. Elles constituent en tout cas des outils extrêmement commodes pour les tâches de développement et de mise au point.

QUE CONTIENNENT LES MÉMOIRES ?

Composants digitaux par excellence, les mémoires ne peuvent contenir que des informations « tout ou rien » : rien que des 1 et des 0, communément appelés bits ou éléments binaires (eb). Rien n'empêche cependant de convenir d'une signification particulière pour des groupes d'un certain nombre de bits, tandis qu'on commence à parler de mémoires analogiques, notamment dans le domaine de la synthèse de son.

L'une des principales caractéristiques d'une mémoire, quel qu'en soit le type, est son « organisation ». Une mémoire de 64 kb, par exemple, peut être organisée en 65 536 mots d'un seul bit, en 8 192 mots de huit bits, en 16 384 mots de quatre bits, etc. (rappelons en effet qu'un « k » vaut 1 024 bits).

L'organisation la plus courante, et de loin, est l'arrangement en mots de huit bits ou octets (*byte* en anglais). Chaque mot peut être considéré comme une case de la mémoire, portant un numéro individuel appelé adresse. Il est donc possible, en spécifiant l'adresse concernée, de lire ou écrire individuellement chacun des octets d'une telle mémoire.

Si le besoin se fait sentir de mots plus « larges » que huit bits, rien n'interdit d'associer plusieurs mémoires. Avec deux mémoires organisées en octets, par exemple, on pourra travailler sur des mots de seize bits.

LES REPRÉSENTATIONS USUELLES

Représentation binaire

La façon la plus simple de définir le contenu d'une mémoire consiste évidemment à représenter directement les bits qu'elle contient.

Un mot de huit bits, notamment, détermine 256 combinaisons différentes de 0 et de 1 (deux à la puissance huit) : de 00000000 à 11111111.

Très parlante dans certains cas particuliers, cette représentation dite binaire (en base deux) est cependant lourde à manier. On emploie donc plus volontiers les notations décimale (base dix) ou hexadécimale (base seize).

Représentation décimale

La représentation décimale est évidemment plus naturelle, habitués que nous sommes à compter en base dix. Un octet est alors représenté par un nombre entier compris entre 0 et 255.

Le passage d'une base à l'autre se fait en attribuant une valeur décimale ou « poids » à chaque bit en fonction de son rang dans l'octet : 1, 2, 4, 8, 16, 32, 64, 128 (les huit premières puissances entières de deux).

La plupart du temps, le « bit de poids faible » ou « bit le moins significatif » est placé à droite, et le « bit de poids fort » ou « bit le plus significatif » à gauche, mais la convention opposée est parfois employée.

Dans ces conditions, le mot binaire 11001011 est normalement équivalent à la valeur décimale $128+64+8+2+1$ soit 203.

Représentation hexadécimale

Les « connaisseurs » préfèrent pour la plupart opérer en hexadécimal, c'est-à-dire en base seize. Chaque octet est scindé en deux groupes de quatre bits appelés *nibbles*, pouvant donc prendre seize valeurs distinctes (deux à la puissance quatre).

Ces seize valeurs (0000 à 1111) sont représentées sous la forme d'un seul caractère : un chiffre de 0 à 9, ou une lettre de A à F, selon les conventions de la figure 2.1.

Par conséquent, un octet se représente tout simplement au moyen de deux caractères : celui du *nibble* de poids fort suivi de celui du *nibble* de poids faible.

Si nous reprenons notre précédent exemple, l'octet 11001011 (203 décimal) peut se scinder en 1100 et 1011, soit C et B. Il s'écrit donc CB en hexadécimal.

Bien entendu, ce principe peut aussi s'appliquer à des mots de plus de huit bits. Un mot de seize bits, par exemple, peut se scinder en quatre *nibbles* et sera donc représentable par quatre caractères hexa (0000 à FFFF).

0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1

Figure 2.1.
Table de conversion
binaire-hexadécimal

	2 ^e chiffre															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Figure 2.2.
Table de
conversion
décimal-
hexadécimal.

Pour éviter tout risque de confusion, on incorpore souvent dans la valeur un signe destiné à préciser la base de numération utilisée : CBh ou 203d, par exemple.

Afin de faciliter les conversions entre octets décimaux et hexadécimaux, la figure 2.2 fournit une table de correspondance à double entrée rassemblant les 256 cas possibles. Son utilisation vous deviendra très vite familière, au moins jusqu'au jour où vous ne jurerez plus que par l'hexa.

Représentation ASCII

Dans les ordinateurs, on se sert couramment d'octets pour représenter les caractères du clavier, de l'écran, ou de l'imprimante (lettres, chiffres, signes divers). La figure 2.3 reproduit le code ASCII (prononcer « aski ») destiné à normaliser cette correspondance.

Il ne concerne que les 128 octets compris entre 0 et 127, car de nombreuses variantes existent pour les suivants (il s'agit le plus souvent de caractères « graphiques »). On remarquera d'ailleurs déjà que les 32 premiers codes remplissent des fonctions particulières. L'octet 7, par exemple, ne déclenche pas l'impression d'un caractère sur l'écran mais l'émission d'un signal sonore (*bell* signifie cloche en anglais). Ces caractères non imprimables sont dits codes de contrôle.

Dec	Hex	Ascii	Dec	Hex	Ascii	Dec	Hex	Ascii	Dec	Hex	Ascii
0	00H	NUL	32	20H	SP	64	40H	@	96	60H	`
1	01H	SOH	33	21H	!	65	41H	A	97	61H	a
2	02H	STX	34	22H	"	66	42H	B	98	62H	b
3	03H	ETX	35	23H	#	67	43H	C	99	63H	c
4	04H	EOT	36	24H	\$	68	44H	D	100	64H	d
5	05H	ENQ	37	25H	%	69	45H	E	101	65H	e
6	06H	ACK	38	26H	&	70	46H	F	102	66H	f
7	07H	BEL	39	27H	'	71	47H	G	103	67H	g
8	08H	BS	40	28H	(	72	48H	H	104	68H	h
9	09H	HT	41	29H	)	73	49H	I	105	69H	i
10	0AH	LF	42	2AH	*	74	4AH	J	106	6AH	j
11	0BH	VT	43	2BH	+	75	4BH	K	107	6BH	k
12	0CH	FF	44	2CH	,	76	4CH	L	108	6CH	l
13	0DH	CR	45	2DH	-	77	4DH	M	109	6DH	m
14	0EH	SO	46	2EH	.	78	4EH	N	110	6EH	n
15	0FH	SI	47	2FH	/	79	4FH	O	111	6FH	o
16	10H	DLE	48	30H	0	80	50H	P	112	70H	p
17	11H	DC1	49	31H	1	81	51H	Q	113	71H	q
18	12H	DC2	50	32H	2	82	52H	R	114	72H	r
19	13H	DC3	51	33H	3	83	53H	S	115	73H	s
20	14H	DC4	52	34H	4	84	54H	T	116	74H	t
21	15H	NAK	53	35H	5	85	55H	U	117	75H	u
22	16H	SYN	54	36H	6	86	56H	V	118	76H	v
23	17H	ETB	55	37H	7	87	57H	W	119	77H	w
24	18H	CAN	56	38H	8	88	58H	X	120	78H	x
25	19H	EM	57	39H	9	89	59H	Y	121	79H	y
26	1AH	SUB	58	3AH	:	90	5AH	Z	122	7AH	z
27	1BH	ESC	59	3BH	;	91	5BH	[	123	7BH	{
28	1CH	FS	60	3CH	<	92	5CH	\	124	7CH	
29	1DH	GS	61	3DH	=	93	5DH	]	125	7DH	}
30	1EH	RS	62	3EH	>	94	5EH	^	126	7EH	~
31	1FH	US	63	3FH	?	95	5FH	_	127	7FH	DEL

Figure 2.3.
Table
d'équivalence
décimal-
hexadécimal-
ASCII.

LA MISE EN FICHIERS

Dès que le contenu d'une mémoire doit être traité sur ordinateur ou simplement archivé sur disquette, on est appelé à le mettre sous la forme de « fichier ».

Au sens informatique du terme, un fichier est tout simplement une suite d'octets enregistrée sur un support quelconque, et que l'on peut donc facilement sauvegarder, copier, transmettre, ou modifier à l'aide de logiciels usuels.

Fichier binaire

Le plus immédiat est évidemment d'enregistrer directement les octets tels qu'ils se présentent dans la mémoire. C'est ce que l'on appelle un fichier binaire.

```
194 128 175 255 174 255 222 254
223 250 210 128 175 255 174 255
222 254 223 250 128 234
```

Figure 2.4.
Exemple de
fichier décimal
(mode texte).

Considérons par exemple la suite de 22 valeurs décimales de la figure 2.4. Si nous les enregistrons dans un fichier sous la forme des 22 octets correspondants, le « listage » du fichier sur écran ou sur imprimante donnera le résultat reproduit à la figure 2.5 (du moins sur un compatible PC).

```
194 * 128 * 175 * 255 * 174 * 255 * 222 * 254
```

Figure 2.5.
Exemple de fichier
binaire (affiché
ASCII).

Il s'agit là de la forme la plus compacte possible, mais il faut avouer qu'elle s'apparente à du « chinois » !

Fichier décimal

La forme la plus explicite pour le commun des mortels est à l'évidence celle de la figure 2.4 (*free format decimal*). Pour la sauvegarder dans ce format, il faut enregistrer un octet par caractère de chaque valeur, encadré par des espaces. Cela occupe donc jusqu'à quatre octets de « texte » par octet de mémoire. En revanche, visualisation en « clair » et modifications sont immédiatement réalisables à l'aide des utilitaires de base dont dispose tout utilisateur de micro-ordinateur.

Fichier hexadécimal

Le même principe pourrait être appliqué à des valeurs hexadécimales n'occupant que deux caractères par octet. Si, de plus, on élimine les espaces intercalaires, on aboutit à la représentation de la figure 2.6, connue sous l'appellation *straight hex*, c'est-à-dire hexadécimal pur.

Figure 2.6.
Exemple de fichier
straight hex.

```
C280AFFFAEFFDEFEDFFAD280AFFFAEFFDEFEDFFA80EA
```

N'occupant que deux octets de texte par octet de mémoire (plus les éventuels caractères de fin de ligne), le *straight hex* représente un bon compromis entre compacité et lisibilité.

Il a cependant été imaginé des variantes « enrichies » de ces fichiers hexadécimaux, dont les plus connues sont Intel Hex et Motorola S, du nom des fabricants de composants qui les ont introduites.

Leur principal avantage est d'inclure des clefs de contrôle permettant de détecter d'éventuelles erreurs lors de la transmission ou de la copie des fichiers, ainsi que des informations précisant à quelles adresses de la mémoire sont destinés les octets. Le gain de place peut être très sensible dans le cas d'une mémoire partiellement remplie.

Fichier Intel Hex

Figure 2.7.
Exemple de fichier
Intel Hex.

La figure 2.7 montre ce que devient notre exemple en Intel Hex.

```
:10000000C280AFFFAEFFDEFEDFFAD280AFFFAEFFF1
:06001000DEFEDFFA80EACB
:00000001FF
```

Chaque ligne du fichier ou « enregistrement » commence par neuf caractères dont la signification est normalisée comme suit :

- caractère n°1 : caractère « deux points » ;
- caractères 2 et 3 : nombre d'octets utiles de l'enregistrement (ici 16, soit 10h en hexadécimal) ;
- caractères 4 et 5 : octet de poids fort de l'adresse d'implantation de cet enregistrement dans la mémoire (en hexadécimal) ;
- caractères 6 et 7 : octet de poids faible de l'adresse (en hexadécimal) ;
- caractères 8 et 9 : inutilisés (00).

Suivent les octets utiles, constitués chacun de deux caractères comme en *straight hex*, deux caractères représentant un octet de *checksum* (clef de contrôle), puis bien sûr un retour chariot et un saut de ligne.

La clef de contrôle est le complément à deux de la somme sur huit bits, sans retenue, de tous les octets utiles plus les deux octets d'adresse et l'octet de longueur.

La dernière ligne est un « enregistrement de fin » ne contenant pas d'octet utile.

Fichier Motorola S

La figure 2.8 reproduit toujours le même fichier, codé cette fois dans le format Motorola S.

```
S1130000C280AFFFAEFFDEFEDFFAD280AFFFAEFFED
S1090010DEFEDFFA80EAC7
S9030000FC
```

Figure 2.8.
Exemple de fichier
Motorola S.

Là encore, chaque ligne contient un certain nombre d'octets que l'on nomme enregistrement, organisé comme suit :

- caractère n°1 : un caractère « S majuscule » ;
- caractère n°2 : identification du type d'enregistrement :
 - 0 = enregistrement d'en-tête ;
 - 1 = enregistrement de données, adresses à 16 bits ;
 - 2 = enregistrement de données, adresses à 24 bits ;
 - 3 = enregistrement de données, adresses à 32 bits ;
 - 7 = fin de fichier, adresses à 32 bits ;
 - 8 = fin de fichier, adresses à 24 bits ;
 - 9 = fin de fichier, adresses à 16 bits.

Suivent un certain nombre de caractères définissant la taille de l'enregistrement (octets utiles, adresse, et clef de contrôle), ici 19 octets soit 13h en hexa, l'adresse d'implantation en mémoire (ici 0000h), les octets utiles codés sur deux caractères, et enfin la clef de contrôle. Celle-ci est le complément à 1 de la somme des bits utiles, de la taille, et de l'adresse.

L'enregistrement se termine toujours par un retour chariot et un saut de ligne. Là encore, la dernière ligne est un enregistrement de fin qui ne contient pas d'octets utiles.

Il est important de connaître ces différentes normes, car beaucoup de programmeurs du commerce attendent les données à programmer sous l'une de ces formes usuelles. Inversement, de nombreux logiciels de développement fournissent le code à programmer dans l'un ou l'autre de ces formats.

Éventuellement, des transcodages peuvent se révéler nécessaires pour faire communiquer entre eux des matériels ou des logiciels n'ayant pas été conçus pour fonctionner ensemble. Des utilitaires relativement simples s'en chargeront fort bien (et il y en a sur la disquette de ce livre !).

LES APPLICATIONS DES MÉMOIRES

Les applications possibles pour les mémoires sont innombrables, selon la signification des bits qu'elles renferment. Que pourraient donc bien signifier nos 22 octets de la figure 2.4 ? Chacun d'eux représente, ne l'oublions pas, huit bits pouvant prendre individuellement la valeur 1 ou 0.

Ces 1 et ces 0 pourraient fort bien correspondre aux états « collé » et « décollé » de huit relais d'un séquenceur. Chaque adresse contiendrait alors les données d'un « pas de programme », pourvu qu'un compteur approprié fasse défiler tour à tour les adresses successives à un rythme adéquat.

Ces 22 octets pourraient tout aussi bien représenter quelques microsecondes de son digitalisé, qu'un simple convertisseur numérique-analogique à résistances n'aurait aucun mal à reproduire par l'intermédiaire du premier amplificateur venu.

Eh bien non ! Il s'agit en fait d'un court programme destiné à un microcontrôleur « 8751 », et servant tout simplement à faire clignoter un voyant.

La figure 2.9 reproduit le code-source de ce programme, c'est-à-dire les instructions en langage assembleur qui ont été traduites en octets directement exécutables par un cross-assembleur (en l'occurrence la version de démonstration du logiciel ECAL, disponible sur le CD-ROM de notre ouvrage *Logiciels PC pour l'électronique* paru dans cette même collection).

Et il ne s'agit là que d'exemples fort classiques ! Programmée astucieusement, une mémoire peut rendre des services des plus variés : remplacer, par exemple, un décodeur sept segments lorsque les modèles standards ne peuvent convenir.

Dans les imprimantes matricielles, c'est aussi une mémoire qui contient la forme de tous les caractères disponibles : il suffirait de la reprogrammer pour imprimer en grec.

Line 1	Col 1	Pos 1	Insert	AutoIndent	A:\NCLI	
File	block	Quick	Error/brEak	Screen	Remember	Options
Delete	place	Ascii	Printer	reBlock	rep Last find	Esc

```
debut:
clr p0.0
mov r7,255
loopa:
mov r6,255
loopb:
djnz r6,loopb
djnz r7,loopa
setb p0.0
mov r7,255
loopc:
mov r6,255
loopd:
djnz r6,loopd
djnz r7,loopc
jmp debut
```

1 Help 2 Diags 3 FPMs 4 ReAssm5 CurSkp6 Zoom 7 Watch 8 Go 9 0 Config

LES LOGICIELS D'AIDE AU DÉVELOPPEMENT

Figure 2.9.
Code source
correspondant
à notre fichier.

Le contenu d'une mémoire peut bien sûr être entièrement défini sur le papier, puis chargé à l'aide d'un programmeur manuel. C'est, en effet, très faisable dans le cadre d'applications ne nécessitant que quelques dizaines d'octets, au plus quelques centaines.

Mais les heureux possesseurs de micro-ordinateurs, et tout particulièrement de compatibles PC, ne soupçonnent pas toujours la puissance latente de leur machine en matière de développement d'applications à base de mémoires.

Certains utilitaires d'usage général peuvent déjà rendre de fiers services lorsqu'il s'agit de visualiser ou de modifier un fichier de type binaire.

La figure 2.10 reproduit par exemple l'écran de travail de PC-TOOLS opérant sur le fichier binaire de la figure 2.5 : on retrouve bien les 22 codes hexa de la figure 2.6, suivis d'octets sans aucun rapport puisque cet utilitaire enchaîne le fichier présent à la suite sur la disquette.

L'exemple de la figure 2.11 illustre une façon très économique de se procurer d'excellents logiciels, sous la forme des disquettes de démonstration (souvent gratuites) des programmeurs du commerce.

Bien souvent, une telle « démo » permet déjà de manipuler ou même de transcoder des fichiers dans les principaux formats que nous avons cités.

Figure 2.10.
Fichier de la
figure 2.5 vu par
PC-TOOLS.

```

PC Tools Deluxe R4.30                               Label Vol=Aucun
-----Service Visualisation/Modification Fichier-----
Voie=A:\*.x
Fichier=CLI.BIN   Secteur Relatif 0000000 Groupe 00680 Sect abs disque 0001370

D placement ----- Codes hex----- Valeur ASCII
0000(0000) C2 80 AF FF AE FF DE FE DF FA D2 80 AF FF AE FF +
0016(0010) DE FE DF FA 80 EA 8C 8D EO AE 9B 8D 9B 8E 8A 9B
0032(0020) 9C D2 EO AA 9B 8E 8D 97 91 92 EO CE D2 CF CA EO
0048(0030) AA 9B 8E 8D 97 91 92 EO CE D2 CF CA EO AA 9B 8E
0064(0040) D0 06 01 0B 52 37 39 37 3E 00 D2 DE B7 AD C8 CA
0080(0050) CA 19 FA F2 1C 2B 58 13 08 AE E1 FC F4 FA 4E 3D
0096(0060) 3E 3E 31 59 48 D5 E8 31 31 BA 06 01 0F 08 91 9C
0112(0070) 5D 3D A1 86 DF 3B 14 83 93 1A BE 58 80 8D 1D 83
0128(0080) DE 95 67 85 0C 85 F4 81 CE 81 E8 90 53 89 6F 74
0144(0090) FE 97 FD 98 3C A0 48 A0 08 96 AF 9F C3 95 28 97
0160(00A0) DA 2D 1B 1D 0B 0E 17 0C 07 53 35 1B 07 60 33 17
0176(00B0) 0D 0D 17 12 19 52 52 52 52 52 52 3B 3D 3F 34 60
0192(00C0) 82 8D 81 87 7E 84 83 13 E9 3A 45 7F 2B 1A DA E7
0208(00D0) E2 E6 30 62 4C 61 BE 5D 6C 21 82 DD 1D 6A AF 33
0224(00E0) 2A EE A1 FE 08 7D 7B 03 20 20 20 20 00 00 00 00
0240(00F0) 00 00 00 00 00 00 BA 4D E2 14 5D 14 C5 0D 00 00

Home=début de fichier/disque End=fin de fichier/disque
ESC=Sortie PgDn=avant PgUp=arrière F1=ASCII/HEX F2=chg n secteur F3=modif

```

Figure 2.11.
Un exemple
d'éditeur de fichiers
«spécial EPROM».

Buffer Co	Addr	Hex	Binary	Bytes
Clear Edi	0:	C2 80 AF FF	11000010 10000000 10101111 11111111	
	4:	AE FF DE FE	10101110 11111111 11011110 11111110	
	8:	DF FA D2 80	11011111 11111010 11010010 10000000	
	C:	AF FF AE FF	10101111 11111111 10101110 11111111	
	10:	DE FE DF FA	11011110 11111110 11011111 11111010	
	14:	80 EA 00 00	10000000 11101010 00000000 00000000	
	18:	00 00 00 00	00000000 00000000 00000000 00000000	
	1C:	00 00 00 00	00000000 00000000 00000000 00000000	
	20:	00 00 00 00	00000000 00000000 00000000 00000000	
	24:	00 00 00 00	00000000 00000000 00000000 00000000	
	28:	00 00 00 00	00000000 00000000 00000000 00000000	
	2C:	00 00 00 00	00000000 00000000 00000000 00000000	
	30:	00 00 00 00	00000000 00000000 00000000 00000000	
	34:	00 00 00 00	00000000 00000000 00000000 00000000	
	38:	00 00 00 00	00000000 00000000 00000000 00000000	
	3C:	00 00 00 00	00000000 00000000 00000000 00000000	
	40:	00 00 00 00	00000000 00000000 00000000 00000000	
	44:	00 00 00 00	00000000 00000000 00000000 00000000	
	48:	00 00 00 00	00000000 00000000 00000000 00000000	
	4C:	00 00 00 00	00000000 00000000 00000000 00000000	
Buffer: C		CURSOR:	O END:	15
Device: N				um: 121EH
Options: L				

F1helpF2radixF3gotoF4reconfigureF5searchF6fillF7copyF8invertF9checksumF10exit

Il s'agit d'une excellente forme de publicité, car, même si l'utilisateur fait ses premières armes en associant ce logiciel à un programmeur « maison », son choix sera vite fait lorsque viendra le moment d'acquiescer un appareil plus élaboré.

Parmi toutes les versions de démonstration qu'il nous a été donné d'essayer, nous avons tout particulièrement apprécié celles des programmeurs MQP, BP Microsystems, et Sprint. Un échantillonnage en est également reproduit sur le CD-ROM accompagnant l'ouvrage précité.

Notre exemple montre comment l'éditeur de fichiers BP Microsystems permet de visualiser et d'éditer un fichier en mode binaire ou hexadécimal (voire même ASCII), quel que soit le format dans lequel il a été sauvegardé !

Lorsqu'il s'agit de travailler sur un programme de microprocesseur, un logiciel assembleur s'impose dès que l'on dépasse les quelques dizaines d'octets qu'il est raisonnablement possible d'assembler à la main.

La version de démonstration de l'assembleur universel ECAL permet de produire du code, sur un compatible PC, pour pratiquement n'importe quel microprocesseur ou microcontrôleur. Cette version limitée ne peut générer plus de 256 octets de code, mais c'est suffisant pour d'intéressantes petites applications tandis qu'il reste très possible de combiner après coup plusieurs de ces courtes routines dans une même mémoire.

Inversement, un désassembleur peut se révéler très utile pour reconstituer le code-source d'un programme à partir du code exécutable. La figure 2.12 reproduit un exemple obtenu à nouveau avec notre fichier de la figure 2.5.

Bien entendu, d'intéressants logiciels de développement peuvent aussi être écrits à la demande, en BASIC ou en Pascal, langages très répandus parmi les utilisateurs de PC. Du fort bon travail peut être accompli par des programmes de quelques dizaines de lignes, et nous ne tarderons pas à le constater.

Figure 2.12.
Notre fichier
traité par un
désassembleur.

AMS 8032 TARGET PROGRAM EDIT SCREEN			
field up	char left	Ctrl=field left	Fgup=prev page
ESC =EXIT	field down	char right	Ctrlfield right
F2 =tree	F3=index	F6=symbolics[OFF]	F7=instructions
		F8=list[OFF]	
		Symbolic entry- type '.' then symbolic name	
0000	C280	CLR	80
0002	AFFF	MOV	R7,FF
0004	AFFF	MOV	R6,FF
0006	DEFE	DJNZ	R6,0006
0008	DFFA	DJNZ	R7,0004
000A	D280	SETB	80
000C	AFFF	MOV	R7,FF
000E	AFFF	MOV	R6,FF
0010	DEFE	DJNZ	R6,0010
0012	DFFA	DJNZ	R7,000E
0014	80EA	SJMP	0000
0016	00	NOP	
0017	00	NOP	
0018	00	NOP	
0019	00	NOP	
001A	00	NOP	
001B	00	NOP	
001C	00	NOP	
PC	MACHINE	LABEL	OPCD OPERAND OPERAND

L'EPROM, MÉMOIRE PROGRAMMABLE PAR EXCELLENCE

Si nous décidons de ne considérer comme composants programmables que ceux susceptibles d'être personnalisés de façon définitive ou au moins durable à l'aide d'appareils relativement simples, alors seules quelques familles de mémoires méritent le qualificatif de programmables.

Les RAM dynamiques ou même statiques ordinaires sont à exclure d'office puisqu'elles perdent leur contenu dès leur mise hors tension.

Les RAM à pile intégrée et les EEPROM se situent dans un cas limite puisqu'elles peuvent conserver leur contenu, hors tension, pendant des années, mais qu'il demeure toujours possible d'y écrire comme dans une RAM. Leur contenu n'est donc pas vraiment figé et reste exposé à des modifications malencontreuses.

Les ROM « masquées », dont le contenu est défini dès la phase de fabrication en usine, ne sont pas modifiables par l'utilisateur. Il faut plutôt les rattacher à la catégorie des composants « sur mesure ».

Les PROM, ou ROM programmables par destruction définitive de fusibles internes, sont faites pour être programmées par l'utilisateur, à l'unité ou en série, à l'aide de programmeurs appropriés. Il s'agit donc, sans aucun doute, de composants programmables. Leur principal avantage est la « robustesse » de leur contenu : il faut beaucoup d'énergie pour faire fondre un fusible, et une fois coupé celui-ci ne peut en aucun cas voir se rétablir sa continuité.

Par contre, il n'est pas possible de les tester à fond en usine, car il faudrait pour ce faire détruire tous les fusibles ! Il est donc considéré comme « normal » que la programmation échoue sur un certain pourcentage de PROM, d'où la nécessité d'une vérification approfondie après programmation.

Les EPROM ou UVPROM sont des ROM programmables par l'utilisateur, et effaçables par simple exposition aux rayons ultraviolets. Si on les produit dans des boîtiers céramique à fenêtre de quartz (matière perméable aux UV), elles sont donc reprogrammables un grand nombre de fois (de l'ordre de quelques centaines).

Logées dans des boîtiers plastique opaques, elles ne peuvent plus être effacées et sont alors dites OTP (*One Time Programmable*). Leur prix rivalise ainsi avec celui des PROM à fusibles, tandis que leur programmation se fait sous une énergie beaucoup plus faible.

Pouvant être programmées, puis effacées en usine avant la mise en boîtier, elles sont testables à 100 % avant d'être livrées.

Leur contenu doit cependant être considéré comme un peu moins « robuste » que celui des PROM à fusibles. Il est en effet normal qu'une EPROM s'efface très lentement sous l'effet du temps, de la chaleur, voire de rayonnements ionisants. On estime cependant qu'il n'y a rien à craindre pendant au moins dix ans dans de bonnes conditions.

Tout cela fait que l'EPROM représente l'un des meilleurs compromis possibles pour l'utilisateur moyen. C'est donc un composant beaucoup plus « populaire » que les PROM à fusibles, et que les mémoires FLASH ne sont pas parvenues à « tuer ». L'amélioration constante des techniques de production se traduit par des EPROM toujours plus performantes, dont certains types récents bénéficient même d'innovations très appréciables.

Les principaux brochages

Bien que la technologie EPROM permette de réaliser des mémoires selon pratiquement n'importe quelle architecture, l'organisation en mots de huit bits est de très loin la plus courante. S'agissant de mémoires statiques, les EPROM bénéficient de la norme dite *Byte-wide* qui vise à standardiser au maximum les brochages des mémoires statiques à mots de huit bits, quel qu'en soit le type : RAM, ROM, PROM, EEPROM, EPROM, FLASH, etc.

Cette standardisation permet, dans certaines limites, d'interchanger sur un même support des mémoires *Byte-wide* de familles et de capacités différentes, ce qui est fort appréciable en pratique !

Le tableau de la figure 2.13 rassemble les brochages des références les plus courantes d'EPROM présentées en boîtier DIL à 24 broches. On ne peut qu'être frappé par le grand nombre de points communs !

Bien évidemment, les broches de masse et d'alimentation V_{CC} (+5 V) sont communes à ces différents modèles. Il s'agit respectivement des broches 12 et 24, disposées en diagonale comme c'est le cas pour bon nombre de circuits logiques courants.

Figure 2.13.
Brochage des principales EPROM à 24 broches.

2716	2732 27C32	27C16	DIL 24 br.	27C16	2732 27C32	2716
A7	A7	A7	1	24	V_{CC}	V_{CC}
A6	A6	A6	2	23	A8	A8
A5	A5	A5	3	22	A9	A9
A4	A4	A4	4	21	V_{PP}	V_{PP}
A3	A3	A3	5	20	$\overline{OE}$	$\overline{OE} / V_{PP}$
A2	A2	A2	6	19	A10	A10
A1	A1	A1	7	18	$\overline{CE} / PGM$	$\overline{CE} / PGM$
A0	A0	A0	8	17	D7	D7
D0	D0	D0	9	16	D6	D6
D1	D1	D1	10	15	D5	D5
D2	D2	D2	11	14	D4	D4
			12	13	D3	D3

© DUNOD - La photocopie non autorisée est un délit.

Les huit lignes de données (D0 à D7) occupent également des emplacements normalisés : les broches 9 à 11, et 13 à 17. C'est par elles que transitent en parallèle, c'est-à-dire sur huit fils à la fois, les données à huit bits que l'on lit ou programme en mémoire.

Les lignes d'adresse sont par contre présentes en nombre différent selon la capacité de la mémoire. La figure 2.14 explique en effet comment N lignes d'adresse définissent 2^N adresses ou « cases », soit 8×2^N bits puisque chaque adresse accueille un octet de huit bits.

Comme en technique numérique les puissances de deux sont reines, l'habitude a été prise de raisonner sur la base d'un préfixe kilo (ou k) valant 1 024 au lieu de 1 000 : 1 024 est en effet la puissance 10 de deux.

Ainsi, une mémoire « 2716 » de 16 kilo-bits contient 2 kilo-octets soit $2\,048 \times 8$ bits, autrement dit 16 384 bits. Cela peut surprendre un néophyte, mais on s'y habitue fort bien !

Figure 2.14.
Architecture de quelques EPROM.

	TYPE	k bits	k octets	Lignes adresse	N = nb lignes adresses	2^N
24 br.	2716	16	2	A ₀ -A ₁₀	11	2048
	2732	32	4	A ₀ -A ₁₁	12	4096
28 br.	2764	64	8	A ₀ -A ₁₂	13	8192
	27128	128	16	A ₀ -A ₁₃	14	16384
	27256	256	32	A ₀ -A ₁₄	15	32768
	27512	512	64	A ₀ -A ₁₅	16	65536

À chaque fois que l'on double la capacité d'une mémoire, il faut ajouter une ligne d'adresse. L'application de la norme *Byte-wide* se traduit par le fait que le brochage précédent est conservé. On attribue à la ligne d'adresse supplémentaire une broche jusque là inutilisée, ou on en libère une en regroupant des fonctions secondaires sur une même broche.

C'est ainsi que la 2716 possède deux broches séparées pour la tension de programmation V_{PP} et pour le signal de lecture /OE. Or, ces deux signaux peuvent très bien se partager une même broche : la tension V_{PP} n'est nécessaire qu'en programmation, alors que /OE n'a besoin d'être au niveau bas qu'en mode lecture.

Sur la 2732, la broche 20 combine donc V_{pp} et /OE, libérant ainsi la broche 21 pour la ligne d'adresse supplémentaire A11.

La figure 2.15 montre comment ce principe se perfectionne au-delà de la capacité de 4 kilo-octets, par passage à un boîtier comportant 28 broches. On arrive ainsi à atteindre la capacité de 64 kilo-octets avec la 27512, tout en maintenant la compatibilité Byte-wide au niveau des 24 broches inférieures du support. Cela signifie que dans un montage bien conçu (programmeur ou circuit utilisateur), on pourra interchanger facilement des EPROM de capacités très variées selon les besoins. Au pire, il faudra simplement déplacer un très petit nombre de cavaliers de commutation.

Lorsque nous avons rédigé la première édition de cet ouvrage (en 1991), le type d'EPROM le plus courant était la 2764, ou sa version CMOS 27C64 qui coûtait déjà presque partout sensiblement moins cher qu'une 2716, malgré une capacité quadruple. Trois ou quatre ans seulement auparavant, nous écrivions en toute bonne foi que la 2716 était l'EPROM à recommander car la plus économique.

Figure 2.15.
Brochage des principales EPROM à 28 broches.

27512 27C512	27256 27C256	27128 27C128	2764 27C64	DIL 28 br.	2764 27C64	27128 27C128	27256 27C256	27512 27C512
A15	V_{pp}	V_{pp}	V_{pp}	1	V_{CC}	V_{CC}	V_{CC}	V_{CC}
A12	A12	A12	A12	2	PGM	PGM	A14	A14
A7	A7	A7	A7	3	N.C.	A13	A13	A13
A6	A6	A6	A6	4	A8	A8	A8	A8
A5	A5	A5	A5	5	A9	A9	A9	A9
A4	A4	A4	A4	6	A11	A11	A11	A11
A3	A3	A3	A3	7	OE	OE	OE	OE/ V_{pp}
A2	A2	A2	A2	8	A10	A10	A10	A10
A1	A1	A1	A1	9	CE	CE	CE	CE
A0	A0	A0	A0	10	D7	D7	D7	D7
D0	D0	D0	D0	11	D6	D6	D6	D6
D1	D1	D1	D1	12	D5	D5	D5	D5
D2	D2	D2	D2	13	D4	D4	D4	D4
				14	D3	D3	D3	D3

Aujourd'hui, le support à 28 broches a progressivement supplanté celui à 24 broches, et rien ne s'oppose à ce que l'on programme seulement le quart d'une 27C64 avec le contenu d'une 2716 : il suffit de câbler au niveau bas les deux lignes d'adresse excédentaires (A11 et A12).

Bien entendu, il a existé des EPROM de capacité inférieure à celle de la 2716, et on rencontre maintenant des EPROM de plusieurs mégabits. On peut toutefois déjà faire une foule de choses avec les 27C256 et les 27C512 qui sont, du moins à l'heure où nous écrivons ces lignes, les EPROM les mieux « placées » dans le commerce de détail.

La figure 2.16 reproduit, à titre de souvenir, le brochage de la 2708 (1 kilo-octet), que nous ne regretterons certes pas puisqu'elle exigeait des tensions spéciales en plus du 5 V, même en lecture.

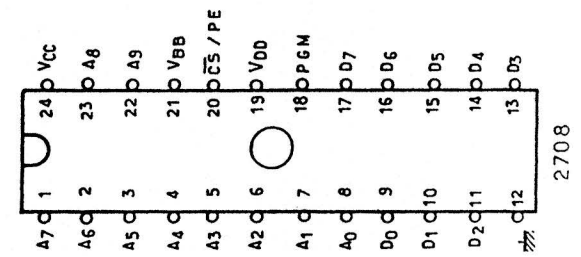
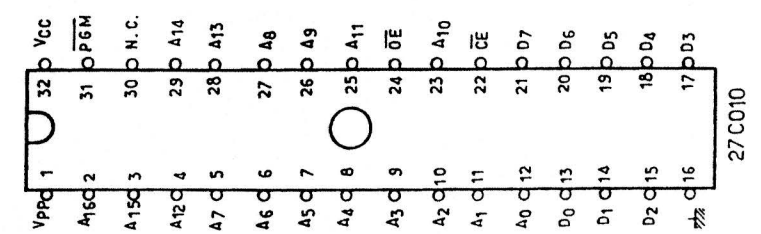


Figure 2.16.
Brochage de la 2708 (1 kilo-octet).

A contrario, la figure 2.17 montre comment la 27C010 atteint le mégabit (1 024 kilo-bits) grâce à quatre nouvelles broches (boîtier DIL à 32 pattes), mais en respectant toujours la compatibilité Byte-wide avec les modèles à 28 et 24 broches.

Figure 2.17.
Brochage de la 27C010 (1 mégabit).



Les algorithmes de programmation

On appelle algorithme de programmation d'un composant programmable la succession d'opérations qu'il faut exécuter pour le programmer correctement : quels signaux appliquer à quelles broches, sous quelles tensions et pendant combien de temps, comment vérifier le succès de la manœuvre, etc.

La technologie même des EPROM suppose que la programmation s'opère sous une tension nettement plus élevée que le 5 V servant en mode lecture. Autrefois de 25 V pour toutes les références, cette tension V_{pp} est passée à 21 V puis à 12,5 V. Il importe de respecter précisément la valeur prescrite par le fabricant (parfois imprimée sur le boîtier mais pas toujours !) sous peine d'abrégé la vie de l'EPROM.

Le V_{pp} est en règle générale appliqué à une broche spécialisée, le V_{cc} restant pour sa part à 5 V ou un peu plus (une tendance récente tend à recommander un V_{cc} de 6 V pendant la programmation, en vue d'une meilleure rétention des données). Notons que V_{pp} doit impérativement être appliqué en même temps que ou après V_{cc} , et être retiré en même temps que ou avant V_{cc} : il y va de la vie de l'EPROM !

En présence de V_{cc} et de V_{pp} , on présente alors l'octet à programmer sur les huit lignes de données, tout en appliquant aux lignes d'adresse les niveaux logiques définissant le numéro de la « case » dans laquelle ces données doivent être inscrites.

La plupart du temps, on programme toutes les adresses dans leur ordre croissant, mais rien n'interdit d'accéder ponctuellement à telle ou telle adresse spécifique.

Une impulsion calibrée (presque toujours négative, mais positive dans le cas particulier de la 2716) est alors appliquée à une broche spécialisée (/PGM). La durée de cette impulsion était autrefois uniformément de 50 ms, d'où une durée de programmation, alors considérée comme acceptable, d'un peu moins de deux minutes pour une 2716.

Dans le cas d'une 27512, c'est presque une heure qu'il faudrait compter dans ces conditions, ce qui est évidemment réhibitore. Les EPROM récentes se contentent donc de 10 ms ou moins, tandis que des artifices ont été imaginés pour accélérer encore les choses (algorithmes rapides).

Afin de fixer les idées, la figure 2.18 indique les durées minimales de programmation des principales références dans le cas d'impulsions de 50, 10, et 1 ms.

Figure 2.18.
Quelques exemples de temps de programmation.

	2716	2732	2764	27128	27256	27512
1 ms	2 s	4 s	8 s	16 s	32 s	1min4s
10 ms	20 s	40 s	1min20s	2min40s	5min20s	10min40s
50 ms	1min42s	3min25s	6min50s	13min40s	27min18s	54min37s

Une lecture de contrôle est en général effectuée à un moment ou à un autre, dans le but de vérifier si la programmation a bien réussi. On place alors la mémoire dans ses conditions normales de lecture, notamment en appliquant un niveau bas sur /CE (sélection de la mémoire) et sur /OE (passage en mode lecture).

À vrai dire, il existe à peu près autant d'algorithmes de programmation que de fabricants de mémoires, si ce n'est autant que de références d'EPROM ! Il n'est pas rare qu'une 2764 de marque X se programme à 25 V et 50 ms (procédé le plus archaïque), et qu'une autre de marque Y se contente de quelques millisecondes sous 12,5 V.

On ne conseillera donc jamais assez de consulter le *data book* de la marque concernée (ou une base de données informatisée) avant de tenter de programmer une EPROM.

En cas de doute (marque effacée), on commencera dans des conditions très « douces » (par exemple 12,5 V et 10 ms), en exécutant aussitôt une lecture de contrôle. S'il y a échec, on forcera progressivement sur la tension et/ou sur la durée (mais ce n'est là qu'un pis-aller).

Les algorithmes rapides

Même avec des impulsions de 10 ms, attendre plus de dix minutes qu'une 27C512 soit programmée semble bien long. Les fabricants d'EPROM ont donc imaginé des procédures plus ou moins compliquées, permettant de gagner du temps.

L'une des plus évidentes, et toujours possible à appliquer, consiste à comparer l'octet à programmer avec le contenu de l'adresse vierge. S'il y a égalité, on passe immédiatement à l'adresse suivante, gagnant ainsi jusqu'à 50 précieuses ms. Dans le cas contraire, bien sûr, on exécute normalement la programmation de l'adresse en question. Ce procédé peut se révéler assez rentable, du fait que bien souvent les EPROM (surtout celles de grande capacité) ne sont que très partiellement remplies.

Une autre astuce consiste à faire une tentative de programmation avec une impulsion très courte (de 0,1 à 1 ms), et à renouveler cette manœuvre autant de fois qu'il le faut pour que la programmation réussisse. Certains algorithmes se contentent ainsi d'appliquer « juste ce qu'il faut », tandis que d'autres ajoutent par sécurité quelques impulsions supplémentaires.

Outre la rapidité, ces procédures présentent l'avantage de ménager au maximum la durée de vie de l'EPROM. Leur mise en œuvre suppose cependant l'utilisation de programmeurs évolués, capables de faire alterner sans cesse des cycles de programmation et de lecture.

Dans tous les cas, un « garde-fou » est prévu, qui limite le nombre de tentatives de programmation : en cas d'échec au bout d'une certaine durée cumulée d'impulsions, la mémoire est déclarée défectueuse.

La figure 2.19 détaille ainsi un algorithme recommandé par NS (National Semiconductor) pour certaines de ses EPROM CMOS. Il correspond à une durée maximale d'impulsion de 10 ms, que l'on peut appliquer d'autorité si on ne dispose que d'un programmeur inadapté à l'algorithme rapide.

La figure 2.20 montre pour sa part comment Atmel recommande d'appliquer une « surprogrammation » de 3 ms pour confirmer le succès de l'opération. On aboutit ainsi à des temps de programmation par adresse compris entre 4 et 25 ms.

Chez Cypress (figure 2.21), on opère avec des impulsions très courtes (100 μ s). Dès que l'octet apparaît programmé, on applique une impulsion supplémentaire dont la durée varie en fonction de ce qu'il a déjà fallu appliquer (c'est là un algorithme dit « adaptatif »).

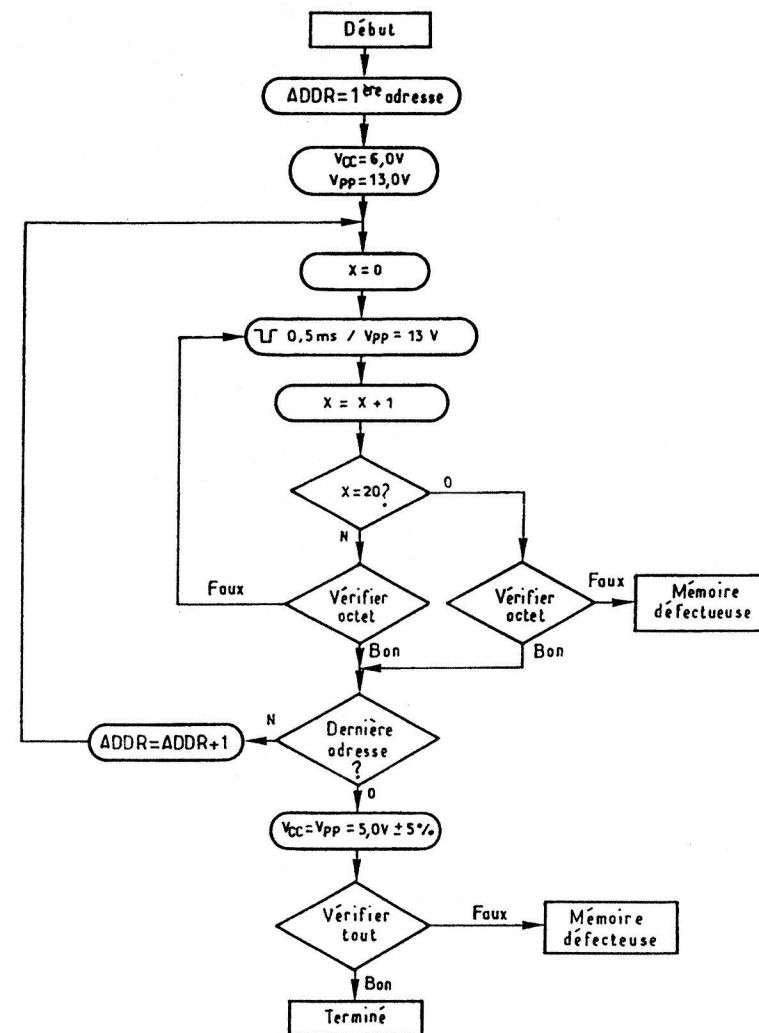


Figure 2.19. Un algorithme rapide préconisé par NS.

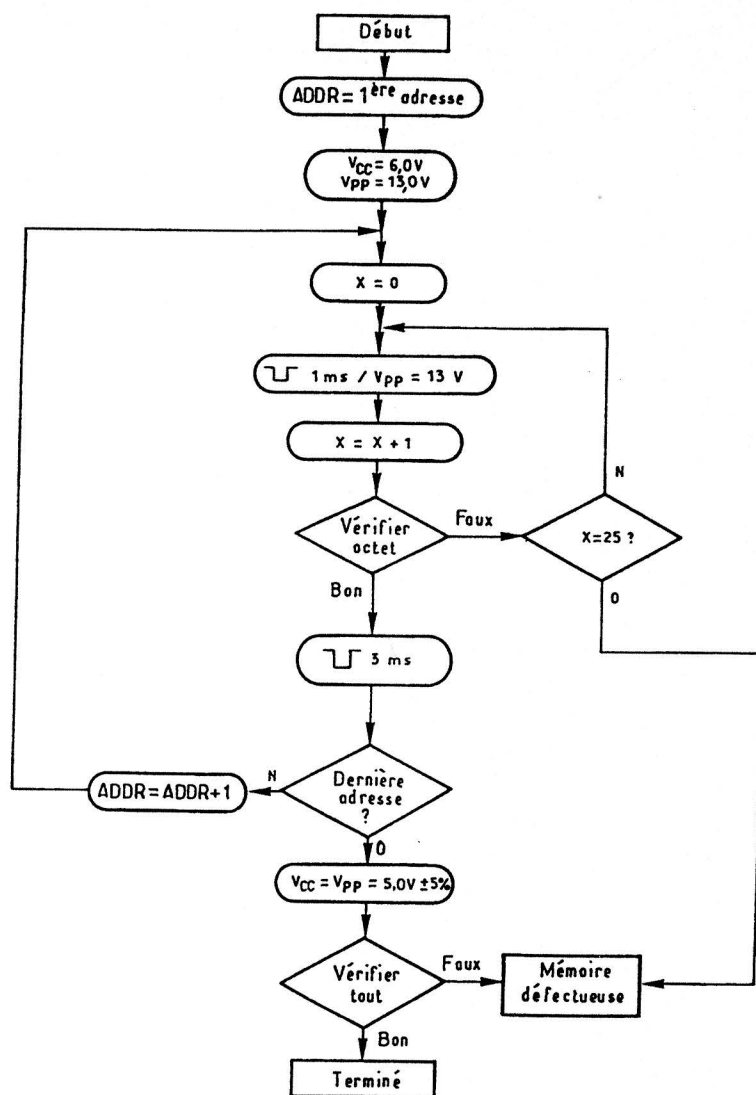


Figure 2.20. Un algorithme rapide suggéré par Atmel.

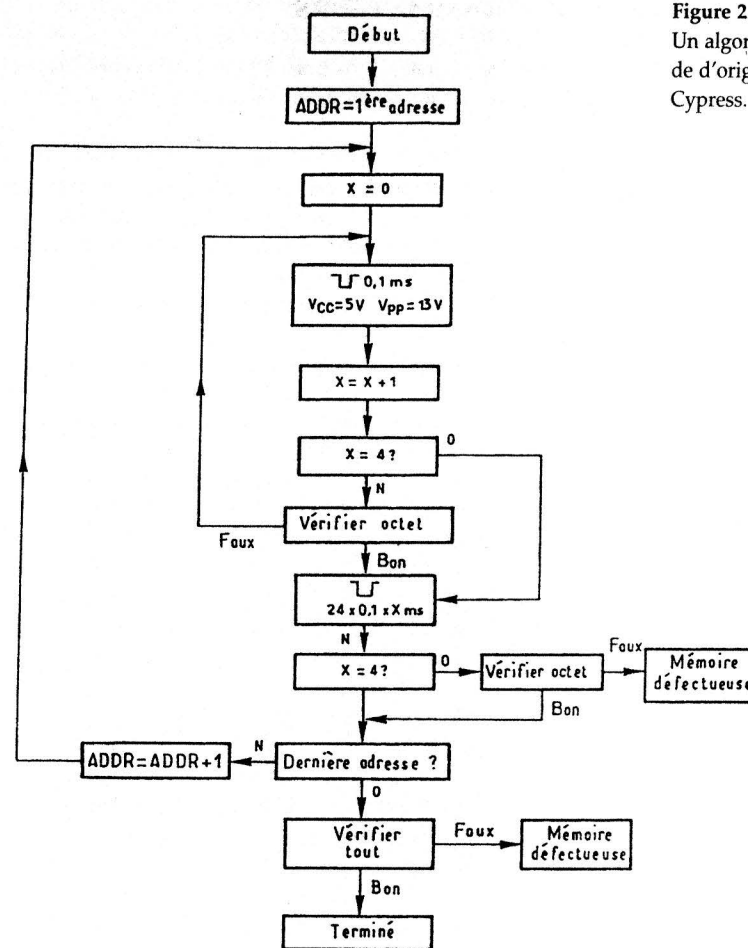


Figure 2.21. Un algorithme rapide d'origine Cypress.

© DUNOD - La photocopie non autorisée est un délit.

Pour un maximum de fiabilité, chaque fabricant recommande vivement d'appliquer à la lettre les algorithmes qu'il préconise, lesquels ont été optimisés en fonction de la technologie employée pour produire les composants.

Pour des applications courantes, on peut cependant se permettre de prendre quelques libertés avec ces procédures. On y gagne la possibilité d'utiliser un programmeur simplifié, infiniment moins coûteux que les modèles « agréés », mais il faut en revanche renoncer à la garantie du fabricant !

LES PROGRAMMATEURS

L'écriture de données dans une EPROM est une opération très particulière, qui nécessite donc un équipement spécifique : le programmeur.

On peut déjà considérer un tel appareil comme une petite machine à fabriquer des circuits intégrés spécifiques, car la programmation d'une EPROM transforme une pièce standard en un composant personnalisé, cela de façon définitive tant que l'on ne procède pas à un effacement volontaire.

Il existe une immense variété de programmeurs, adaptés aux besoins les plus divers. La situation est en effet fort différente selon qu'il s'agit de programmer un seul prototype à plusieurs reprises au cours d'un travail de développement, ou de produire des séries de pièces identiques en un minimum de temps.

Les duplicateurs

Passons rapidement sur cette catégorie de programmeurs, qui sert essentiellement à produire un grand nombre de pièces identiques à un échantillon servant de modèle.

Comme le rendement de production est primordial, ces appareils exécutent en général plusieurs copies en même temps (par exemple huit). Par ailleurs, une vérification systématique des pièces produites est indispensable afin d'éviter que d'éventuels échecs de programmation ne se répercutent plus loin dans la chaîne de fabrication.

Ces programmeurs « de production » sont la plupart du temps autonomes : ils se suffisent à eux-mêmes en ne nécessitant qu'un simple branchement au secteur pour fonctionner.

Les programmeurs manuels

Un programmeur manuel est lui aussi un appareil autonome, dépourvu de tout moyen de communication informatique. Pouvant généralement faire office de duplicateur simplifié (une seule copie à la fois), il sert surtout à programmer facilement n'importe quelle donnée à n'importe quelle adresse grâce à une batterie d'interrupteurs et de voyants.

C'est très pratique en phase de développement, car on peut pratiquement construire le contenu d'une EPROM bit par bit,

au fur et à mesure des progrès de la mise au point du montage. Seuls les uns peuvent toutefois être transformés en zéros, l'opération inverse exigeant l'effacement global de la mémoire et donc la perte de tout le travail déjà effectué !

Il est bien évident qu'il serait excessivement fastidieux de programmer de cette façon les 8 192 adresses d'une simple 27C64, sans même parler des EPROM de capacités supérieures ! Ce mode opératoire reste par contre envisageable jusqu'à quelques centaines d'octets, ce qui couvre déjà un certain nombre d'applications intéressantes.

Les programmeurs pour micro-ordinateurs

Le micro-ordinateur (et plus particulièrement le « compatible PC ») est devenu un outil très courant, peu coûteux, et facilement disponible. Son utilisation en tant que programmeur d'EPROM, grâce à des extensions spécialisées, apporte une formidable puissance de traitement et une aide considérable au développement d'applications, aussi complexes soient-elles.

Le contenu d'une EPROM de capacité quelconque peut en effet être représenté par un « fichier » enregistré sur disquette, et donc extrêmement facile à copier et à communiquer.

Un tel fichier peut aussi être créé, consulté, ou modifié à l'aide d'une foule de logiciels existants ou écrits pour la circonstance.

Des bases de données sur disquette permettent de déterminer les valeurs de programmation et les algorithmes applicables à pratiquement toutes les références de toutes les marques, et donc de procéder à la programmation dans les meilleures conditions possibles.

Il faut distinguer deux grandes familles de programmeurs adaptables sur micro-ordinateur : les « externes » et les « internes ».

Les programmeurs externes

Un programmeur externe est un appareil plus ou moins autonome, mais doté de possibilités de communication avec un micro-ordinateur. La liaison se fait la plupart du temps par un port série « RS 232 ».

Les fichiers représentant le contenu des EPROM à programmer sont souvent « téléchargés » par l'ordinateur dans une mémoire vive intégrée au programmeur, qui s'occupe ensuite du

transfert dans une ou plusieurs EPROM. Inversement, une EPROM placée sur le programmeur peut être lue, et son contenu rapatrié vers l'ordinateur qui en fait un fichier disque.

Un dialogue plus élaboré est généralement possible entre les deux appareils, l'ordinateur pouvant envoyer toutes sortes d'ordres codés au programmeur, et recevoir en retour de multiples informations.

L'inconvénient le plus souvent avancé de cette disposition est que la liaison série, relativement lente, constitue un « goulot d'étranglement » de nature à augmenter la durée des opérations. Pour télécharger à 19 200 bauds les 65 536 bits d'une 2764 pleine, il ne faut pourtant qu'à peine cinq secondes, délai qu'il convient de comparer avec la durée de programmation proprement dite.

Bref, techniquement fondée, cette objection n'a somme toute que peu d'incidence pratique. En revanche, l'installation de l'appareil se limite au branchement d'une fiche dans un connecteur externe vacant, ou libéré pour la circonstance (on n'a guère besoin d'une souris ou d'un modem pour programmer des EPROM ...).

On appréciera souvent de ne pas avoir besoin d'un slot, soit que le PC utilisé n'en possède pas, soit qu'ils soient déjà tous occupés (c'est bien plus fréquent qu'on pourrait le penser !).

Certains de ces programmeurs externes peuvent également être employés seuls, grâce à un clavier et à un afficheur incorporés. Pour faire des copies, par exemple, mais aussi pour opérer des modifications d'ampleur limitée ou pour programmer directement un nombre raisonnable d'adresses.

Un appareil particulièrement représentatif est le programmeur « S4 » du fabricant britannique Dataman : de la taille d'une grosse calculatrice, il fonctionne sur accumulateurs incorporés. On peut donc l'emmener partout, et ne le connecter au PC que lorsque cela s'impose vraiment. Même en mode autonome, ses possibilités particulièrement étendues en font un authentique système de développement, puissant et souple. Il possède même un cordon d'émulation lui permettant de prendre la place d'une EPROM dans un montage en cours de mise au point !

Une autre approche consiste à utiliser la prise d'imprimante Centronics pour brancher certains programmeurs, libérant ainsi la RS 232 pour d'autres usages. S'agissant d'une prise « parallèle », la vitesse de transmission est plus importante.

Les programmeurs internes

Beaucoup de programmeurs nécessitent l'installation d'une carte spécifique dans un slot vacant du PC, carte à laquelle le boîtier recevant les EPROM vient se raccorder par un connecteur spécifique.

Il existe même des modèles de bas de gamme qui intègrent directement les supports d'EPROM sur la carte : inaccessibilité garantie !

Communiquant directement avec le cœur même de l'ordinateur, ces appareils sont évidemment les plus rapides. Par contre, ils sont pratiquement toujours inutilisables sans leur ordinateur « hôte ».

Le dialogue très riche qui peut s'établir avec l'ordinateur facilite la mise en œuvre de fonctions plus puissantes que celles des programmeurs externes.

D'une façon générale, les possibilités d'un programmeur interne bien conçu ne dépendent pratiquement que du logiciel : la prise en compte de nouvelles références ou de nouveaux algorithmes ne nécessite la plupart du temps que l'envoi d'une disquette de mise à jour ou un téléchargement par modem, du moins tant que le fabricant de l'appareil existe ...

On ne saurait donc trop recommander de faire confiance à des fournisseurs solidement implantés et jouissant d'une bonne réputation, et de se méfier des abonnements « à vie » ou des mises à jour gratuites promises par certains illustres inconnus !

Les kits et programmeurs « maison »

Les programmeurs du commerce sont souvent de splendides outils, dotés de vastes possibilités et (en principe) d'une grande fiabilité.

Beaucoup de fabricants de tels appareils entretiennent des rapports privilégiés avec les fournisseurs de composants, et ont donc connaissance du détail des algorithmes de programmation souvent cachés à l'utilisateur final.

Certains sont même officiellement « agréés » par les principaux fabricants de composants programmables.

Leur principal inconvénient est d'être chers : de quelques milliers à plusieurs dizaines de milliers de francs !

Une approche plus économique consiste à construire soi-même son programmeur, soit à partir d'un kit du commerce, soit par ses propres moyens.

Il existe des kits fort valables qui, pour quelques centaines de francs, permettent de programmer les EPROM courantes dans de bonnes conditions. Soit de façon entièrement manuelle, soit par copie d'un modèle, soit même avec l'assistance d'un micro-ordinateur.

D'une façon générale, ils ne peuvent appliquer aucun algorithme spécifique, ni programmer d'autres composants que les EPROM les plus répandues.

Nous fournirons dans la suite de cet ouvrage les plans de programmeurs particulièrement simples, capables de programmer différentes familles de composants au prix d'un investissement dérisoire.

Il ne pourront cependant traiter que des composants dont l'algorithme de programmation est public. Fort heureusement, il en existe suffisamment pour résoudre déjà bien des problèmes pratiques, tandis que ce sont justement ceux que l'on trouve facilement dans le commerce de détail à des prix abordables !

L'EFFACEMENT DES EPROM

Bien que divers autres facteurs puissent entraîner l'évanouissement progressif du contenu d'une EPROM, c'est par exposition à un rayonnement ultraviolet que l'on efface pratiquement ces composants, remettant ainsi tous leurs bits à 1.

Il est donc évidemment nécessaire que le boîtier de la mémoire soit muni d'un hublot perméable à la longueur d'onde de 3 000 à 4 000 angströms pour laquelle l'effacement est le plus efficace.

On livre donc généralement les EPROM dans des boîtiers en céramique équipés d'une fenêtre en quartz, qui sont hélas particulièrement coûteux à fabriquer.

Les EPROM destinées à la production en série n'ayant souvent nul besoin d'être effacées, il a été imaginé de recourir aussi à des boîtiers en plastique, opaques mais bien plus économiques.

Ces EPROM ineffaçables sont dites « OTP » et sont en principe programmables une seule fois. Il reste cependant toujours possible de transformer en 0 les bits demeurés à 1, ce qui autorise dans certains cas une programmation en plusieurs fois.

Il faut une certaine « dose » de rayonnement pour effacer une EPROM de façon sûre, mais il existe aussi une irradiation cumulée au-delà de laquelle la mémoire doit être considérée comme « usée ». Il est donc important de bien régler l'exposition.

La durée d'effacement dépend de multiples facteurs : nature, degré de vieillissement et propreté de la source d'UV, propreté de la fenêtre de l'EPROM, distance de l'EPROM à la source, etc.

À condition de n'être pas pressé, la lumière du soleil peut d'ailleurs suffire d'où la nécessité de coiffer d'une étiquette opaque (et vraiment opaque !) la fenêtre des EPROM programmées de façon définitive.

La source UV la plus utilisée pour l'effacement des composants programmables de toutes sortes est le tube fluorescent « germicide », largement employé pour la stérilisation de toutes sortes de produits et pour la désinfection de locaux.

Ces tubes existent dans les puissances classiques des tubes fluorescents d'éclairage et utilisent les mêmes accessoires (connecteurs, ballasts, starters). On préfère la plupart du temps employer le plus petit modèle possible, mais le TG 15 ou TUV 15, de 15 W et 44 cm de long, est relativement populaire chez les amateurs.

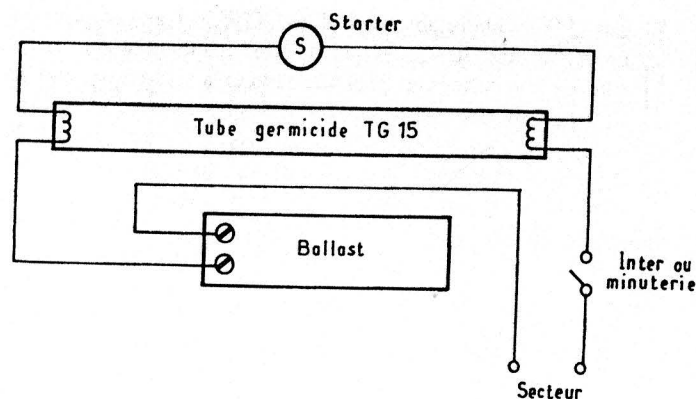
À signaler aussi un modèle sans ballast, le TUV 6. Muni d'une simple douille à vis, il s'alimente directement en 220 V et ne consomme que 6 W.

Ces modèles particulièrement adaptés à l'usage qui nous intéresse sont tenus en stock par de nombreux revendeurs de composants, qui proposent également des kits complets.

Une autre approche consiste à se fournir chez un électricien, et à réaliser le schéma de la figure 2.22.

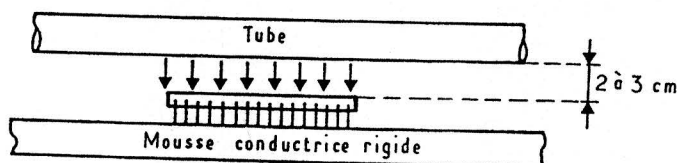
La durée d'irradiation nécessaire étant de l'ordre de cinq à trente minutes, il est intéressant de prévoir une minuterie éliminant tout risque de surexposition. Un modèle mécanique très simple fait parfaitement l'affaire.

Figure 2.22.
Schéma électrique
d'un effaceur
d'EPROM.



Dans tous les cas, il faut réaliser un boîtier permettant de disposer les composants à effacer de telle sorte que leur fenêtre soit écartée de 2 à 3 cm du tube, de préférence en les piquant dans de la mousse antistatique comme le montre la figure 2.23.

Figure 2.23.
Disposition des
EPROM par rapport
au tube UV.



Dans la mesure du possible, le tube sera placé au-dessus des composants à effacer et non en dessous, afin de limiter leur échauffement. Il faudra de toute façon combiner une aération satisfaisante du tube avec une bonne protection contre les fuites d'UV, ce rayonnement étant nocif pour les yeux et la peau. Bien souvent, un système à tiroir est le plus indiqué. L'effaceur terminé (ou acheté tout fait) il reste à l'étalonner. Une EPROM préalablement programmée sera exposée par tranches de cinq minutes suivies d'un « test de virginité », opération disponible sur la majorité des programmeurs. On appliquera par la suite une durée double de celle ayant suffi pour effacer complètement la mémoire, mais il sera prudent de procéder à un réétalonnage de temps à autre (mettons deux fois par an).

Terminons en signalant l'existence de « pistolets effaceurs » munis d'un tube flash spécial émettant dans l'ultraviolet. Ces appareils sont capables d'effacer une EPROM, sans même la retirer de son support, en quelques dizaines de secondes.

LES ÉMULATEURS D'EPROM ET LES RAM À PILE

Les mémoires RAM réalisées en technologie CMOS consomment suffisamment peu pour pouvoir être alimentées pendant des années sur une toute petite pile au lithium. Leur contenu devient alors « non volatil », mais reste modifiable dans les conditions normales d'écriture en RAM : on peut surcharger aussi bien un 1 par un 0 qu'un 0 par un 1. L'effacement global devient donc inutile !

Il a été développé des composants qui rassemblent dans un boîtier DIP spécial une RAM CMOS, une pile au lithium, et un circuit de supervision d'alimentation assurant la sécurité des données. Ce sont les ZEROPOWER de SGS Thomson, les NVSRAM de Benchmarq ou Dallas, et les « émulateurs d'EPROM » de Greenwich Instruments, pour ne citer que les plus connus. En lecture, ces mémoires sont compatibles à 100 % avec les EPROM de capacité équivalente. On peut donc les enficher dans n'importe quel montage pour simuler ou « émuler » une EPROM. Pas question par contre de les programmer comme des EPROM, c'est-à-dire en « haute tension » : il faut les alimenter uniquement en 5 V, et appliquer une impulsion négative sur une broche /WR en présence de l'adresse et des données.

Quelques centaines de nanosecondes suffisent, ce qui fait qu'en une fraction de seconde on peut programmer une telle mémoire avec plusieurs centaines de kilo-bits. C'est l'idéal pour les tâches de mise au point qui nécessitent de fréquentes modifications du contenu de la mémoire.

Une fois le contenu de la RAM à pile définitivement fixé, rien n'est plus simple que de le recopier dans une EPROM.

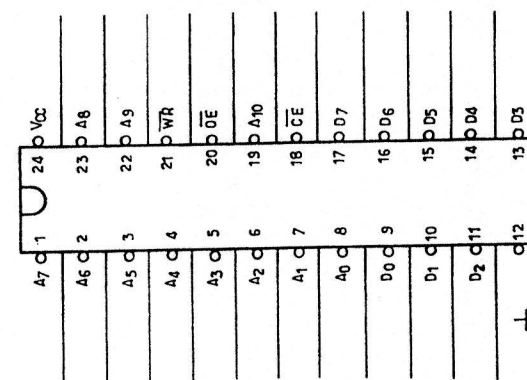
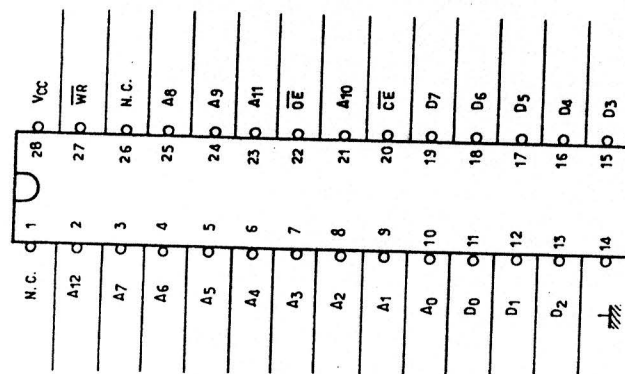


Figure 2.24.
Brochage de la
48Z02 (2 Ko).

La figure 2.24 reproduit le brochage de la MK 48Z02 de SGS THOMSON (2 kilo-octets, compatible avec la 2716), et la figure 2.25 celui de la MK 48Z08 (8 kilo-octets, compatible avec la 2764).

Figure 2.25.
Brochage de la
48Z08.



La GR 281 de Greenwich Instruments est équivalente à la 48Z02, et la GR 881 à la 48Z08.

À part la broche d'écriture /WR, les broches /OE et /CE s'utilisent de la même façon que celles des EPROM correspondantes.

LES MÉMOIRES EEPROM SÉRIE

La technologie EEPROM (ou E2PROM) permet de réaliser des mémoires présentant un attrait certain pour de nombreuses applications, réunissant dans une certaine mesure les avantages des EPROM et des RAM.

À côté de toute une famille de produits à accès parallèle, généralement conformes au standard Bytewise, les EEPROM « série » s'imposent irrésistiblement dans des domaines très variés, des périphériques d'ordinateurs à la télévision, en passant évidemment par les cartes à puce.

Les capacités couramment disponibles vont de 128 bits à 16 kilo-bits et plus, rejoignant donc celles des petites EPROM parallèles.

Les protocoles de communication sont généralement assez classiques, à commencer par des dérivés de l'I²C.

Le concept EEPROM

Comme son nom l'indique, une EEPROM (*Electrically Erasable Programmable Read Only Memory*) est en principe une mémoire « morte » comparable à une EPROM, mais effaçable électriquement et non plus aux ultraviolets.

Mais les EEPROM présentent l'immense avantage, par rapport aux EPROM ou même aux mémoires FLASH, de pouvoir être effacées de façon partielle, et pas forcément globale. L'effacement des modèles récents est même « transparent », en ce sens qu'il est effectué automatiquement avant l'écriture de toute nouvelle donnée. Dans ces conditions, on peut s'interroger sur ce qui différencie une EEPROM d'une RAM « non volatile », par exemple à pile lithium incorporée.

Tout d'abord, le temps d'accès en écriture. Comme une EPROM, l'EEPROM enregistre ses données au moyen d'un processus « haute tension » relativement lent. Même si la génération de la tension de programmation par une « pompe de charge » interne à l'EEPROM peut faire oublier cette parenté, quelques millisecondes restent nécessaires pour l'écriture de chaque mot de données.

En revanche, il suffit d'inhiber la haute tension pour interdire toute écriture, volontaire ou accidentelle. Une EEPROM peut donc offrir davantage de sécurité qu'une RAM à pile.

Des solutions élégantes ont été imaginées pour contourner autant que possible le handicap de la durée d'écriture. Dans les NOVRAM de Xicor, par exemple, chaque cellule EEPROM est « doublée » par une cellule de RAM. Tout se passe donc comme si on était en présence d'une RAM, mais qui peut être sauvegardée en bloc dans une EEPROM en vue d'un rechargement futur. Les transferts peuvent être commandés électriquement, par logiciel, ou automatiquement lors de la coupure ou de l'établissement de la tension d'alimentation.

Autre différence essentielle entre RAM et EEPROM, l'endurance limitée de ces dernières aux cycles d'écriture : selon les marques, des erreurs d'écriture commencent à apparaître après dix mille à un million d'effacements par cellule, alors que les RAM tolèrent des écritures en nombre illimité.

Déjà très supérieurs à ceux des EPROM, ces chiffres peuvent être améliorés par des techniques de correction d'erreurs ou de redondance, applicables soit dans le montage utilisateur, soit dans la mémoire elle-même. L'utilisation d'un code de

Hamming, par exemple, peut permettre d'augmenter d'environ six fois la vie utile d'une EEPROM, puisque le défaut commence pratiquement toujours par n'affecter qu'un seul bit par mot. Une relecture de contrôle après chaque écriture est donc évidemment à conseiller pour toute application exigeant une totale intégrité des données confiées à la mémoire.

Des progrès sensibles ont été accomplis au niveau de la durée de rétention des données par les EEPROM, souvent bien supérieure à celle des EPROM et des RAM à pile : au moins dix ans, et jusqu'à un siècle selon les marques !

Même si les choses ont évolué ces dernières années, une certaine méfiance reste souvent de rigueur quant à la fiabilité à long terme des piles et de leurs connexions, notamment en environnement rude. Rien de tel évidemment avec les EEPROM ! Et ajoutons que l'on imagine encore mal l'installation d'une pile dans un boîtier DIP à huit broches, ou dans un boîtier CMS.

Les EEPROM à accès série

C'est bien connu, chaque broche supplémentaire sur un circuit intégré coûte cher : au niveau du prix du composant lui-même, bien sûr, mais aussi en termes de surface et de complexité du circuit imprimé qui l'accueille.

Même si les EEPROM coûtent sensiblement plus cher que les EPROM de capacité équivalente, le principe « série » permet de faire de substantielles économies.

Bien que forcément pénalisante au plan de la vitesse des échanges, la mise en œuvre de liaisons série à deux ou trois fils seulement entre circuits intégrés se révèle finalement avantageuse pour bon nombre d'usages. Le succès confirmé du bus I2C est là pour en témoigner !

Ajoutons que certaines applications ne seraient même pas envisageables autrement, à commencer par les cartes à puce à huit ou même six contacts seulement...

L'apparente complexité des échanges de données nécessaires a d'abord nui au succès des EEPROM à accès série, bien des développeurs s'entêtant à utiliser des mémoires parallèles Byte-wide, même pour quelques octets seulement.

Émettre ou recevoir des données série sur un port de microcontrôleur est maintenant devenu tellement trivial que cette

réticence fait heureusement place à l'enthousiasme. Comment rester insensible au charme d'un petit boîtier à huit broches qui peut conserver pendant cent ans autant de données qu'une 2716, s'alimente en 5 V seulement (ou même 2,5 V), dont on peut lire ou modifier tout ou partie du contenu par quelques instructions émises sur un simple bus à deux ou trois fils, et qui possède éventuellement des sécurités de protection en écriture ? De plus en plus nombreuses sont donc les marques qui proposent des EEPROM série, et c'est bien normal compte tenu du succès grandissant de ces composants.

Les performances sont cependant assez sensiblement différentes d'une marque à l'autre, notamment en matière de tension d'alimentation, d'endurance, et de rétention des données. Typiquement, l'éventail va de 5 V/10 000 écritures/10 ans à 2,5 V/1 000 000 d'écritures/100 ans, avec toutes les combinaisons intermédiaires. Un choix s'impose donc selon les applications envisagées, les prix pratiqués, et les fonctionnalités souhaitées, car bien des surprises sont possibles à ce niveau !

Une même référence (par exemple la très courante 93C06) peut correspondre à des produits très nettement différents d'une marque à l'autre : brochage, organisation, et même jeu d'instructions. La plus grande prudence est donc de rigueur lors du choix de « secondes sources », de préférence dès le début de la conception du projet.

Nous n'avons évidemment pas la prétention de cataloguer ici toutes les références, ni tous les fabricants d'EEPROM série !

Deux grandes familles semblent appelées à s'imposer en concurrence : les EEPROM à bus « deux fils » style I2C, et celles à bus « trois fils » style Microwire ou SPI (bus dédié à l'origine au 6805 et au 68HC11).

I2C et Microwire sont en réalité deux marques déposées, la première par Philips et la seconde par NS. Seuls leurs propriétaires ou les fabricants en ayant acquis la licence peuvent s'en prévaloir officiellement, mais on rencontre bien évidemment de multiples imitations, plus ou moins fidèles...

Le cas de l'I2C est particulièrement net. Énormément de marques ont acquis la licence et peuvent donc utiliser ce sigle, à condition de respecter les spécifications techniques qui y sont liées. Leurs composants doivent normalement fonctionner sans problème dans n'importe quel environnement I2C en bonne et due forme.

Certaines firmes n'ont par contre pas souhaité acquérir la licence I2C, et ne peuvent donc pas vendre leurs produits sous ce label. Malgré la dénomination SCL et SDA de leurs lignes de bus, leur compatibilité avec les spécifications officielles I2C n'est souvent qu'approximative, et parfois même inexistante. Il peut donc en résulter des conflits avec d'autres composants si on cherche à les utiliser dans un véritable environnement I2C.

Mais il s'agit tout de même d'excellents produits, capables de rendre de grands services moyennant la mise en œuvre d'un protocole de communication très proche de l'I2C, mais parfois légèrement différent (et échappant donc en principe à la licence...).

Les EEPROM I2C ou similaires

Figure 2.26.
Panorama des
principales
EEPROM I2C.

Le tableau de la figure 2.26 rassemble les références les plus répandues d'EEPROM I2C ou apparentées.

I²C et similaires (bus 2 fils)

ORGANISATION		SGS-THOMSON	XICOR	ATMEL	MICROCHIP
16 × 8			24C00 24001		
128 × 8		24C01	24C01 24C01A 24012	24C01	85C72 24C01A 24LC01 24LH01
256 × 8		25C02A 24C02A	24C02 24022	24C02	85C82 24C02A 24LC02 24LH02
2 × 256 × 8		25C04 24C04	24C04	24C04	85C92 24C04A 24LC04 24LH04
4 × 256 × 8		24C08	24C08	24C08	
8 × 256 × 8			24C16 24164	24C16	24C16 24LC16 24LH16

Compte tenu des spécifications I2C, toutes ces mémoires sont organisées en mots de huit bits, tandis que les capacités supérieures à 256 octets sont couramment scindées en autant de « pages » de 256 octets que nécessaire. La capacité la plus cou-

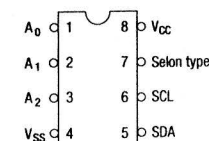
rante est donc évidemment 256 octets, tandis qu'on dépasse maintenant, grâce à différents artifices logiciels, la limite théorique de 2 kilo-octets (8 blocs de 256), soit 16 kilo-bits.

La figure 2.27 résume pour sa part ce qu'il est essentiel de connaître sur le plan du brochage.

Quatre broches sont en principe communes à presque toutes les versions de quasiment toutes les marques :

- (4) V_{ss} (masse) ;
- (5) SDA (données) ;
- (6) SCL (horloge) ;
- (8) V_{cc} (alimentation).

Figure 2.27.
Brochage des
EEPROM I2C.



En principe, les broches 1, 2, et 3 sont affectées aux trois bits de poids faible de l'adresse I2C du composant (A0, A1, A2), les quatre bits de poids fort étant 1010, et le bit des unités l'indicateur de lecture-écriture (R/W).

Cela permet notamment de faire coexister jusqu'à huit composants identiques sur un même bus, chacun bénéficiant d'une adresse distincte.

Tel n'est pourtant pas le cas des 24C00, 24001, et 24C01 de XICOR, qui ne possèdent pas d'adresse « de composant » et attendent directement une adresse d'octet : leurs broches 1 à 3 ne sont donc pas connectées.

Cette organisation (non-conforme à la spécification I2C) permet une simplification du dialogue, mais exclut la coexistence de plusieurs composants sur le même bus. Cela peut toutefois suffire dans des cas simples !

La 24C01A (Xicor ou Microchip), par contre, possède les trois broches A0, A1, A2 et les gère bien au niveau de son « adresse d'esclave », tandis que la 24C01 d'Atmel les possède mais ne les gère pas...

Les choses se corsent encore au niveau de la broche 7 : non connectée sur les 24C00, 24001, 24C01, 24012 Xicor, et 85C72 Microchip, elle doit être mise à la masse dans le cas des 24C01 Atmel et des 24C01A ou 24LC01 Microchip.

Sur la 24C01A Xicor, par contre, la mise à la masse de la broche 7 autorise l'écriture alors que sa mise au niveau haut l'interdit en bloquant le générateur interne de tension de programmation !

Sur les 24C02A et 25C02A SGS-Thomson, cette broche 7 permet de choisir le nombre d'octets pouvant être enchaînés dans une même opération d'écriture (4 ou 8).

Sur la 24C02 Xicor et la 24LC02 Microchip, la broche 7 permet d'autoriser ou d'interdire l'écriture dans toute la mémoire, tandis que sur la 24C02A Microchip cette protection n'intéresse que la moitié supérieure de la mémoire.

Encore plus fort : à partir de 512×8 bits, la capacité est scindée en blocs de 256 octets, qu'il faut pouvoir sélectionner dans l'adresse « d'esclave ». On se sert pour cela d'un, deux, ou trois des bits affectés jusqu'ici aux broches A0, A1, et A2.

Dans le cas d'une mémoire de 4 kilo-bits ($2 \times 256 \times 8$ bits), seules les broches A1 et A2 influent sur l'adresse du composant. Il ne peut donc coexister au maximum que quatre de ces mémoires sur le même bus, mais le bit P0 ainsi libéré permet de choisir l'une ou l'autre des deux « pages » de 256 octets.

Du coup, la broche 1 (anciennement A0) se trouve libérée : dans la plupart des cas, il faut la relier à la masse, mais on peut la laisser en l'air sur la 24C04 Atmel.

Sur les 24C04 et 25C04 (version 2,5 V) de SGS-Thomson, la broche 1 sert à mettre en action un système assez sophistiqué de protection en écriture, que nous ne décrivons que succinctement.

Quand la broche 1 est au niveau haut, le dernier octet de la mémoire (adresse FFh dans la page P0 = 1) fait office de registre de protection qui détermine, en fait, la frontière entre la zone protégée en écriture et la zone accessible librement en écriture comme en lecture.

Si la broche 1 est à la masse, toute la mémoire est accessible normalement en écriture comme en lecture, ce qui permet notamment de modifier le contenu du registre de protection, situé lui-même dans la zone protégée.

Sur les mémoires de 8 kilo-bits (4 pages de 256 octets), seule la broche A2 participe à la fixation de l'adresse du composant, libérant ainsi deux bits P0 et P1 pour la sélection logicielle de pages. Sur les mémoires de 16 kilo-bits, enfin, l'adresse n'est pas modifiable par commutation car il faut trois bits (P0, P1, P2) pour différencier les 8 pages de 256 octets.

En fait, il est pratiquement équivalent de monter un seul boîtier de 16 kilo-bits sur un bus I2C ou d'y associer huit boîtiers de 2 kilo-bits : l'adressage se fait de la même façon, mais on y gagne en compacité.

La 24164 Xicor occupe une place à part dans cette famille. Au lieu de rester inutilisées, les broches 1, 2, et 3 servent à modifier la partie jusqu'alors immuable de l'adresse du composant : seul le bit de poids fort de celle-ci reste fixé à 1, les trois autres dépendant des niveaux appliqués sur les broches S0, /S1, et S2. Si ces trois broches sont mises à la masse, alors on retrouve l'adresse 1010 habituellement utilisée pour les mémoires I2C.

Il convient de remarquer que cette adresse 1010 a été allouée sur le plan international par le « Comité I2C » aux mémoires de tous types. Utiliser une autre adresse fait courir inévitablement un risque de conflit avec d'autres composants I2C éventuellement présents sur le même bus.

Bien que cette particularité de la 24164 puisse se révéler utile dans certaines applications, il semble clair que ce composant est tout à fait « hors la loi » en matière de normalisation I2C ! Mais au fait, pourquoi pas si son fabricant ne se réclame pas du sigle I2C ?

Sans rentrer dans des détails réputés connus de tout utilisateur du bus I2C, rappelons que les transferts de données y font appel à un nombre limitatif de conditions logiques sur les lignes SCL et SDA :

- _ bus non occupé (SCL et SDA au niveau haut) ;
- _ condition de début ou Start (front descendant sur SDA alors que SCL est au niveau haut) ;
- _ condition de fin ou Stop (front montant sur SDA alors que SCL est au niveau haut) ;
- _ donnée valide (SDA stable pendant que SCL est au niveau haut) ;
- _ accusé de réception ou ACK (SDA forcée au niveau bas pendant le neuvième bit d'un transfert) ;
- _ écriture (bit de poids faible de l'adresse à 0) ;
- _ lecture (bit de poids faible de l'adresse à 1).

Les EEPROM Microwire ou similaires

Le tableau de la figure 2.28 rassemble les références les plus courantes d'EEPROM Microwire ou compatibles.

Initialement optimisées pour fonctionner avec les microcontrôleurs COP de NS, ces mémoires sont habituellement organisées en « registres » de 16 bits.

Certains modèles se prêtent cependant à être reconfigurés en un nombre double de registres de 8 bits.

Figure 2.28.
Panorama des
principales EEPROM
Microwire.

MICROWIRE et similaires (bus 3 fils)

ORGANISATION	NS	SGS-THOMSON	XICOR	ATMEL	MICROCHIP
16 x 16	9306/7 93C06 93CS06	93C06	24C44 24C45 (NOVRAM)		93C06
32 x 8		93C06			
32 x 16	93CS26				
64 x 16	9346 93CS46	93C46 93CS46 93CS47		93C46	93C46 59C11 93LC46
128 x 8		93C46		93C46	59C11 93LC46
128 x 16	93CS56	93CS56 93CS57		93C56	93C56 93LC56
256 x 8				93C56	93C56 93LC56
256 x 16	93CS66			93C66	93C66 93LC66
512 x 8				93C66	93C66 93LC66

Les capacités demeurent plus modestes que celles des versions I2C, l'un des types les plus populaires étant le 16x16 bits (soit seulement 32 octets), particulièrement bon marché.

À noter que ces 256 bits (la capacité d'une puce de télécarte) suffisent amplement pour bien des applications, notamment d'identification : une simple 9306 peut par exemple déjà héberger seize codes offrant chacun 65 536 combinaisons possibles !

Appartiennent aussi à cette catégorie les NOVRAM 24C44 et 24C45 autostore de Xicor, produits particulièrement originaux et riches d'applications.

Le bus Microwire est généralement présenté comme un bus « trois fils », hors masse et alimentation : sélection de boîtier (CS), horloge (SK), et données.

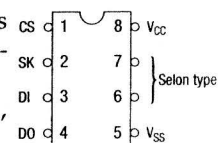
Deux fils sont toutefois prévus pour les données : une entrée (DI) et une sortie (DO). Cela évoque bien sûr un bus « 4 fils », mais en pratique on réunit couramment DI et DO sur un seul et même fil, même si cela suppose quelques précautions pour éviter les conflits.

Le brochage de ces composants est pratiquement standardisé, du moins en ce qui concerne six des huit broches des boîtiers comme le montre la figure 2.29.

Figure 2.29.
Brochage des
EEPROM
Microwire.

C'est au niveau des broches 6 et 7 que de multiples variantes peuvent être recensées d'une référence à l'autre, et même parfois d'une marque à l'autre.

Sur les modèles les plus simples (NMC 9306, NMC 9346, etc.), ces deux broches sont tout simplement inutilisées.



Sur les NOVRAM de Xicor, elles servent à gérer matériellement les processus de sauvegarde et de rappel des données entre la RAM et l'EEPROM. Mais on rencontre aussi les affectations suivantes, qui correspondent chacune à un perfectionnement spécifique :

- _ BPE (*Bulk Programming Enable*) permet d'autoriser ou non les écritures et effacements en bloc de l'ensemble de la mémoire ;
- _ PRE (*Protect Register Enable*) permet d'autoriser ou non l'accès au registre de protection, dont le contenu définit l'adresse du premier registre de la zone devant être protégée en écriture ;
- _ PE (*Program Enable*) permet d'autoriser ou non l'écriture et l'effacement ;
- _ ORG (*ORGanisation*) sert à fixer l'architecture de la mémoire en registres de 16 ou de 8 bits ;
- _ TEST est une broche servant à des tests en usine, et devant être utilisée conformément aux exigences du fabricant (en général à laisser en l'air ou à mettre à la masse).

Différentes combinaisons de ces broches sont possibles selon les références et les marques, de même que différents jeux d'instructions. En effet, le dialogue avec les mémoires Microwire est plus compliqué (mais aussi plus riche) que ce qui se fait habituellement en I2C : la mémoire attend sur sa broche DO des messages composés d'un bit de start (à 1), suivi d'un code opération (opcode) définissant l'instruction à exécuter, et s'il y a lieu d'une adresse de registre puis de données.

S'il s'agit d'une instruction de lecture, les bits lus apparaîtront sur la broche DO dès la fin du message envoyé sur DI (s'il est valide !), le tout bien sûr au rythme imposé par l'horloge SK.

Évidemment, le format de ces échanges diffère plus ou moins d'une référence à l'autre, notamment en fonction du nombre de bits composant les adresses. Il est donc impératif de se procurer les spécifications détaillées de chaque composant avec lequel on envisage de travailler.

Pour une même référence, les opcodes peuvent même être quelque peu différents d'une marque à l'autre (par exemple entre NS et Microchip) !

1	Introduction aux composants programmables	9
2	Les mémoires programmables	13

3 LES RÉSEAUX LOGIQUES PROGRAMMABLES

Notion de réseau logique programmable	58
Les architectures de base	59
Les PAL, réseaux logiques programmables par excellence	62
Les PAL les plus populaires	65
Les PAL universels et les GAL	71
Que contiennent les PAL et les GAL ?	75
Les logiciels d'aide au développement	75
Les programmeurs	81
Les « grands » EPLD	82
Les pLSI et ispLSI de Lattice	84

4	Les microcontrôleurs	95
5	Réalisation de programmeurs	111
6	La disquette du livre	155

NOTION DE RÉSEAU LOGIQUE PROGRAMMABLE

Les réseaux logiques programmables (*Programmable Logic Devices* ou PLD) exploitent des technologies comparables à celles des mémoires programmables, mais dans des buts très différents.

À la base, un réseau logique programmable est un circuit intégré contenant un grand nombre de fonctions logiques de base (surtout des portes et des bascules), dont l'utilisateur peut définir l'interconnexion à volonté, tant au niveau du schéma interne réalisé qu'à celui de l'affectation des broches d'entrée-sortie. Il est donc clair que cette technique permet la réalisation de circuits logiques hautement personnalisés.

Lorsque cette personnalisation se fait par masquage lors de la fabrication du circuit (*gate arrays*), le procédé n'est viable que pour des productions en grande série tandis que toute erreur ou modification coûte très cher. De plus, un long délai est à prévoir entre la fin de l'étude et la mise à disposition des premiers échantillons d'évaluation.

Il est infiniment plus confortable et plus souple que l'utilisateur puisse personnaliser lui-même ses réseaux logiques, à l'unité si nécessaire, à l'aide d'un équipement simple et peu coûteux comparable à un programmeur d'EPRM. Et que demander de plus si le réseau logique ainsi programmé peut même être effacé aux UV ou électriquement, puis reprogrammé ? Peut-être une protection contre la copie du circuit par un « bit de sécurité »...

Eh bien, tout cela existe, de même que de puissants logiciels tournant sur compatibles PC et parfois gratuits, qui prennent en charge la programmation des réseaux en fonction de la description des fonctions logiques devant être obtenues, ainsi que leur vérification par « simulation logique ».

Bref, il est aujourd'hui possible d'écrire ce qui ressemble à un « programme », puis d'en faire un circuit intégré spécifique en quelques étapes simples et rapides ! Bien que certains fabricants s'ingénient à décourager cette pratique, il est même envisageable de construire soi-même le programmeur nécessaire, ce qui peut se révéler étonnamment simple dans certains cas... Pour les constructeurs, ce principe offre de multiples avantages : gain en compacité et en simplicité des circuits imprimés, accélération du développement de nouveaux produits, amélioration

de la fiabilité, diminution du nombre de pièces à stocker, augmentation de la compétitivité, protection contre le « pillage » des schémas.

L'acheteur profite pour sa part, directement ou indirectement, de bon nombre de ces avantages mais doit rester conscient du fait qu'il se trouve « pieds et poings liés » au niveau de la maintenance : il n'aura pas accès aux parties les plus significatives du schéma et ne pourra se procurer des réseaux programmés de rechange qu'auprès du fabricant, si, et seulement si, tel est son bon plaisir...

LES ARCHITECTURES DE BASE

Bien que chaque fabricant de réseaux programmables s'efforce d'apporter son propre « plus » par rapport à la concurrence, on retrouve bien souvent les mêmes architectures de base, au point de pouvoir parler de compatibilité dans bon nombre de cas. Toutes proportions gardées, une mémoire PROM ou EPROM peut déjà être considérée comme un réseau logique programmable : en mode « lecture », elle possède des lignes d'entrée (le bus d'adresses) et des lignes de sortie (le bus de données). À chaque combinaison d'états des entrées peut correspondre n'importe quelle combinaison d'états des sorties : il suffit de programmer les données voulues aux adresses voulues.

C'est ainsi que l'on peut, par exemple, programmer une EPROM pour lui conférer le comportement d'un décodeur « 7 segments », notamment lorsque la « table de vérité » des modèles standards ne donne pas satisfaction. On ne se sert alors que de quatre lignes d'adresse, de sept lignes de données, et on ne programme que seize adresses, soit une infime partie de la capacité disponible.

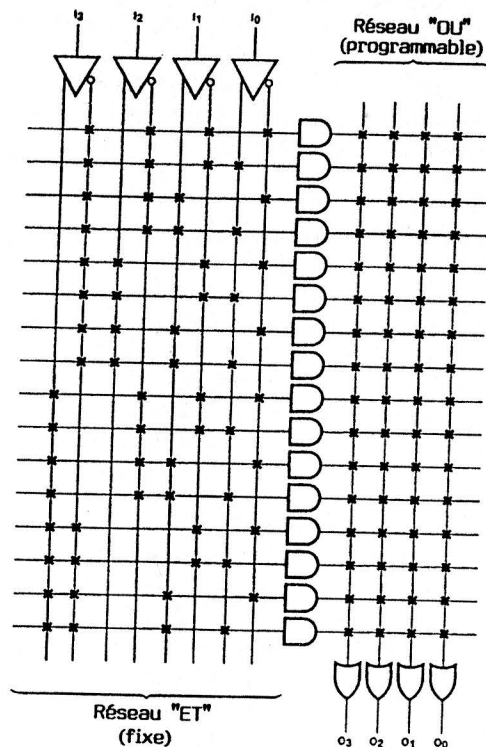
D'un point de vue logique, une EPROM peut être assimilée à un réseau FIXE de portes ET (une par adresse disponible, munie d'autant d'entrées qu'il y a de lignes d'adresse), attaquant un réseau programmable de portes OU (une par ligne de données disponible, munie d'autant d'entrées qu'il y a d'adresses).

Une simple 27C64, par exemple, est équivalente à 8 192 portes ET à 13 entrées, suivies de 8 portes OU à 8192 entrées, avec entre les deux 65 536 points d'interconnexion programmables individuellement comme ouverts ou fermés.

Cela donnerait presque le vertige !

Pour dessiner le schéma correspondant, des conventions simplificatrices sont indispensables. La figure 3.1 représente ainsi l'exemple d'une EPROM fictive équipée de seulement quatre lignes d'adresses et quatre lignes de données.

Figure 3.1.
Organisation logique
d'une EPROM.



Cela correspond déjà à 4 portes OU à 16 entrées chacune, et à 16 portes ET à 4 entrées chacune, le tout réuni par une matrice d'interconnexion à 64 points. Toutes les entrées d'une même porte sont représentées par une seule ligne, verticale pour les OU et horizontale pour les ET. Elles croisent à angle droit les lignes susceptibles de leur être raccordées : sorties des portes ET pour les portes OU, et sorties des tampons d'entrée pour les portes ET.

Bien entendu, les entrées des portes ET peuvent recevoir chacune des lignes d'entrée et aussi leurs compléments.

Toute connexion entre une verticale et une horizontale est matérialisée par une croix ou un point. Ces connexions sont évidemment fixes au niveau des entrées des portes ET (elles

définissent l'architecture de la mémoire), mais programmables par l'utilisateur entre les portes ET et les portes OU. Sur notre exemple, toutes les liaisons possibles sont établies, ce qui correspond au cas d'une EPROM vierge (tous les bits à 1).

La programmation consiste en quelque sorte à détruire sélectivement ces « fusibles », toute ouverture d'un point de connexion correspondant à l'écriture d'un 0 au bon endroit de la mémoire. Les EPROM peuvent donc servir à réaliser toute une variété de circuits purement combinatoires, mais souvent au prix d'une sous-utilisation des possibilités du composant. Par ailleurs, la détermination des données à programmer n'est pas toujours des plus faciles (il faut 8 192 octets pour gérer entièrement 13 lignes d'entrée et 8 lignes de sortie...)

La figure 3.2 illustre une approche différente du problème : l'architecture « FPLA » (Field Programmable Logic Array), dans laquelle le réseau de portes ET est lui aussi programmable.

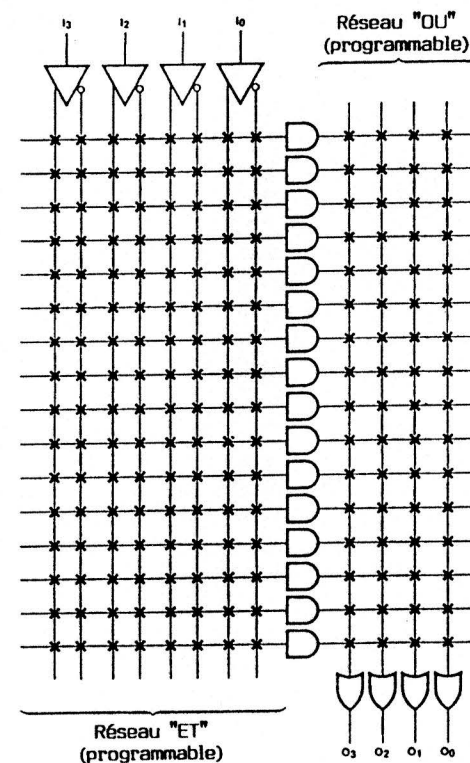


Figure 3.2.
Organisation
logique d'un FPLA.

Cette disposition offre évidemment plus de souplesse, ce qui permet une meilleure utilisation des ressources internes. En revanche, le travail de développement reste assez ardu. À partir de ces architectures de base, les fabricants de circuits intégrés ont imaginé d'innombrables variantes, plus ou moins compatibles entre elles, qui élargissent encore le champ d'application de la logique programmable. Une amélioration courante consiste à intercaler une macrocellule elle-même programmable en amont de chaque broche de sortie.

Mais certains réseaux logiques programmables plus évolués ne rappellent que de très loin leurs aînés : les fabricants rivalisent maintenant d'imagination pour intégrer toutes sortes de fonctions logiques personnalisables à l'intérieur de matrices d'interconnexion, elles-mêmes programmables par l'utilisateur. Proposés par des marques telles qu'Altera, Atmel, Intel, ICT, Xilinx, Actel, Lattice, AMD, ces composants de haut de gamme tendent à se démocratiser d'une façon fort intéressante.

LES PAL, RÉSEAUX LOGIQUES PROGRAMMABLES PAR EXCELLENCE

Le concept PAL (*Programmable Array Logic*) n'est pas particulièrement récent puisqu'il a été introduit par Monolithic Memories (MMI) vers les années soixante-dix. Il a depuis été repris par de multiples fabricants, chacun apportant à l'occasion quelques améliorations de son cru, parfois décisives. L'architecture interne des circuits PAL découle de la constatation selon laquelle toute fonction logique combinatoire peut se ramener à une somme de produits (rappelons que le terme « somme » désigne conventionnellement le OU logique, et le terme « produit » le ET logique).

Des outils classiques tels que le théorème de Morgan, la méthode de Quine Mc Cluskey ou les tableaux de Karnaugh permettent d'obtenir de telles équations booléennes *minimisées* à partir d'équations brutes, ou de tables de vérité directement issues de l'étude du système logique à réaliser.

Un réseau PAL se compose par conséquent des couches logiques suivantes :

- un certain nombre de broches d'entrée, équipée chacune d'un inverseur permettant de disposer à la fois du signal d'entrée et de son complément ;
- un réseau programmable de portes ET à entrées multiples, chaque entrée de chaque porte ET pouvant être connectée à

- n'importe quel signal d'entrée ou à son complément à travers une matrice programmable ;
- un réseau fixe de portes OU recevant chacune les sorties d'un certain nombre des portes ET ;
- éventuellement, un certain nombre de circuits de sortie intercalés entre les sorties des portes OU et les broches de sortie du PAL : il peut s'agir d'inverseurs, de tampons 3 états, ou de bascules D. Dans ce dernier cas, le PAL peut servir à réaliser non seulement des systèmes combinatoires, mais aussi des systèmes séquentiels relativement complexes tels que compteurs et/ou registres ;
- généralement, un certain nombre de circuits de « rétroaction » réinjectant les niveaux de sortie dans la matrice programmable, afin qu'ils puissent être traités comme des signaux d'entrée.

La figure 3.3 montre que cette disposition est en fait exactement l'inverse du cas des EPROM, étudié plus haut.

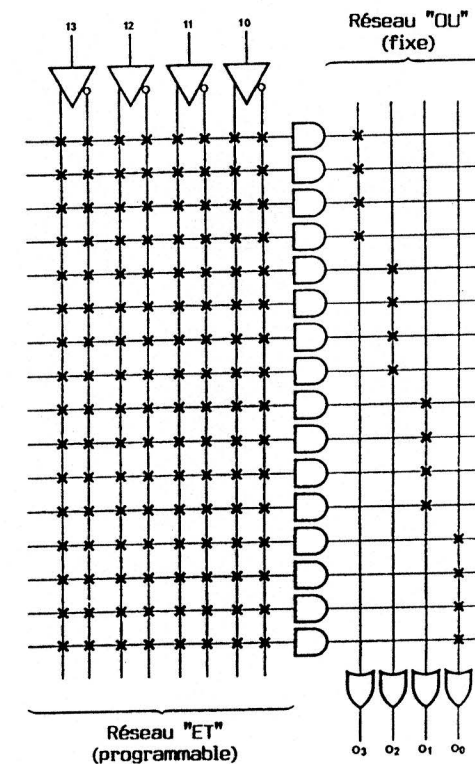


Figure 3.3.
Organisation logique
d'un PAL.

Chaque porte ET possède autant d'entrées que la matrice compte de verticales (typiquement 32, parfois davantage), mais chaque entrée ne rejoint la verticale qui lui est affectée que par l'intermédiaire d'un fusible.

Initialement intact, celui-ci peut être détruit de l'extérieur lors des opérations de programmation. De tels fusibles à destruction irréversible équipaient réellement les PAL « bipolaires », mais ils sont maintenant remplacés par des transistors MOS à effet de champ : au lieu de brûler physiquement un fusible, on place simplement une charge sur la porte flottante du transistor, exactement comme dans une EPROM. Bien que capable de subsister des années, cette charge peut être rapidement éliminée soit électriquement, soit par exposition aux UV si toutefois le PAL est présenté dans un boîtier à fenêtre. Nous sommes alors en présence de PAL effaçables et reprogrammables, appartenant à la famille des EPLD (*Erasable PLD*). C'est l'idéal pour les tâches de développement qui nécessitent en général quelques tâtonnements !

Présentés en boîtiers plastique opaques mais bien plus économiques, les EPLD-UV perdent leur faculté d'effacement : comme les EPROM, on les appelle alors OTP (*One Time Programmable*).

Les sorties des portes ET se nomment « termes de produits » et sont regroupées (typiquement par huit) au niveau des portes OU attaquant les sorties du PAL. C'est là que s'effectue la somme de produits finale.

Un PAL simple comporte couramment 8 portes OU à 8 entrées, soit 64 termes de produits issus de 64 portes ET à 32 entrées chacune, sans oublier les 8 inverseurs d'entrée et les 8 circuits de sortie (inverseurs, tampons, bascules, etc.). Cela correspond à 2 048 fusibles.

Les schémas internes de tels composants seraient évidemment inextricables si on ne leur appliquait la notation particulière dont nous avons déjà fait usage dans le cas des EPROM. On représente ainsi l'ensemble des entrées d'une porte ET par une seule ligne horizontale de la matrice programmable, qui coupe bien sûr la totalité des lignes verticales correspondant aux termes d'entrée ou de rétroaction, directs et complémentés.

Chaque entrée reliée à un terme d'entrée (autrement dit chaque fusible intact) est matérialisée par une croix ou un point à l'intersection de l'horizontale et de la verticale concernées.

Dans la pratique, les PAL les plus simples disposent fréquemment de huit entrées et huit sorties, capacité déjà intéressante mais permettant l'utilisation d'un boîtier à vingt broches seulement (PAL dits de la « série 20 », la plus courante avant la « série 24 » à vingt-quatre broches autorisant davantage d'entrées et de sorties).

La complexité du schéma augmente vite avec le nombre d'entrées et de sorties : la figure 3.3 montre ce que serait la matrice d'un PAL à quatre entrées, quatre sorties, et seize termes de produits, mais nous n'allons pas nous limiter à cela !

LES PAL LES PLUS POPULAIRES

L'architecture que nous venons de définir admet évidemment de multiples variantes. C'est heureux, car il ne faudrait pas imaginer que n'importe quel PAL permettrait de résoudre n'importe quel problème de logique.

Lorsqu'on a besoin de nombreuses entrées et sorties, il est bien entendu possible d'associer plusieurs boîtiers PAL, tout en conservant un gain de quatre ou cinq fois par rapport à l'emploi de circuits logiques des séries 74 ou 4000. Une limite de complexité est cependant introduite par le nombre fini de termes de produits disponibles à l'entrée de chaque porte OU.

Par rapport aux PAL, les FPLA (*Field Programmable Logic Arrays*) offrent plus de souplesse, puisqu'ils possèdent une seconde matrice programmable entre les portes ET et les portes OU, au prix il est vrai d'une complexité accrue au niveau du développement des applications.

On arrive à obtenir une souplesse voisine avec les PAL en multipliant les versions de base, chacune offrant une variante différente de l'architecture commune.

La référence de chaque PAL, outre des informations d'intérêt secondaire, contient une « racine » capable de renseigner sur l'organisation interne du composant.

Le PAL 16L8, par exemple, dispose d'une matrice à seize entrées (soit trente deux verticales avec les complémentés) et de huit sorties actives à l'état bas (active Low).

Même capacité en ce qui concerne le PAL 16R8, à ceci près que les sorties sont équipées de bascules (*Registered*).

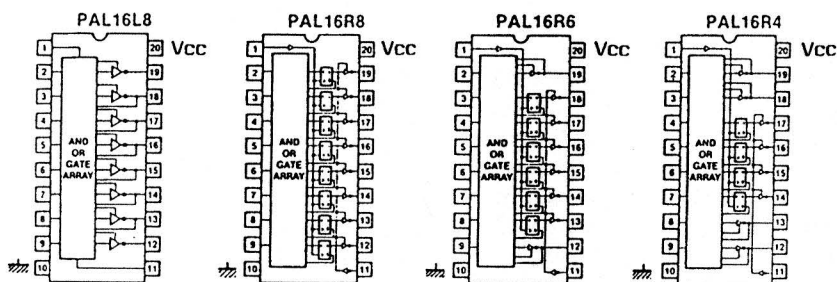
Des configurations intermédiaires sont disponibles, qui pana-
chent les sorties munies de tampons et de bascules. Le PAL
16R4, par exemple, ne possède que quatre bascules tandis que
le PAL 16R6 en offre six, et donc seulement deux tampons.

La lettre H identifie les sorties combinatoires actives au niveau
haut (active *High*), le X celles munies d'une sortie OU eXclusif,
le P celles à polarité de sortie Programmable.

Pour commencer à découvrir les PAL, il nous semble préfé-
rable de se limiter aux PAL 16L8 et 16R8, plus éventuellement
les 16R6 et 16R4, références les plus représentatives et sans
doute aussi les plus pédagogiques.

La figure 3.4 reproduit leur brochage en mode « utilisation »,
totalement différent de celui utilisé en mode programmation.

Figure 3.4.
Brochages des PAL
les plus courants

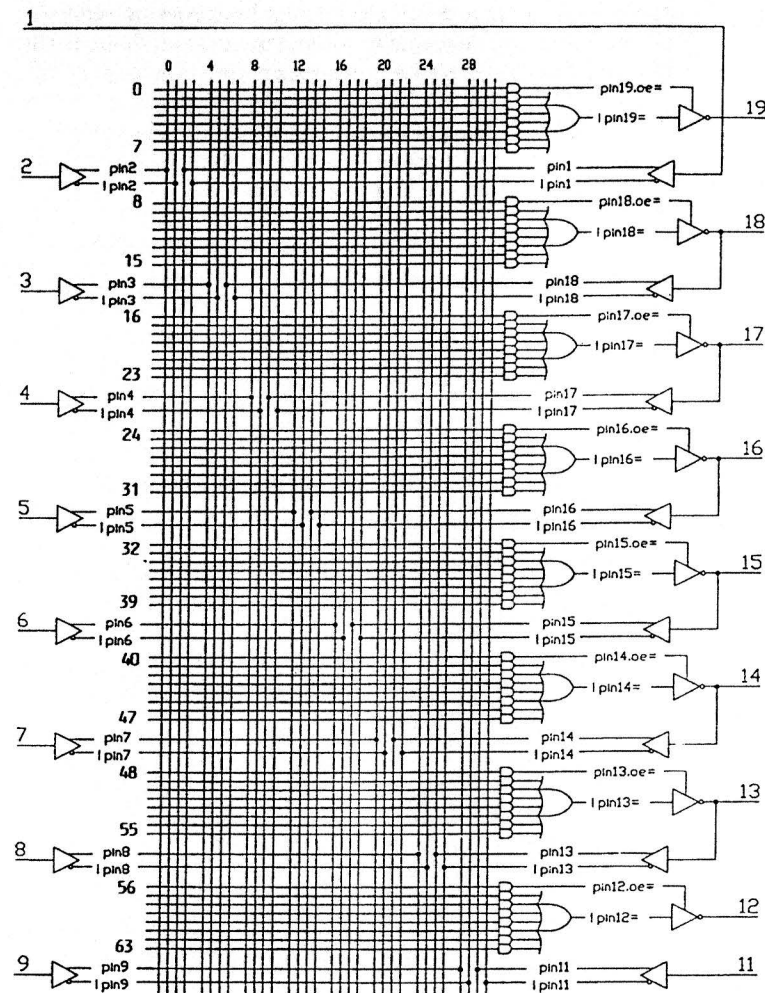


Le PAL16L8

La figure 3.5 reproduit l'intégralité du schéma interne du PAL
16L8 (sans distinction de marque), en respectant les conven-
tions de notation déjà définies. Toutefois, pour des raisons de
commodité, les croix correspondant aux fusibles intacts n'ont
pas été portées. En effet, la fonction logique réalisée étant défi-
nie par les fusibles qu'on laisse intacts, il est plus pratique
d'ajouter des croix que d'en effacer !

Ce PAL possède dix broches d'entrée (1 à 9 plus 11), la n°11
servant souvent d'entrée de validation des tampons de sortie
bien que ce ne soit nullement une obligation.

À ces dix entrées correspondent donc, en ajoutant leurs com-
pléments, vingt verticales de la matrice programmable sur les
trente deux disponibles (0, 1, 2, 3, 4, 5, 8, 9, 12, 13, 16, 17, 20, 21,
24, 25, 28, 29, 30, 31).



© DUNOD - La photocopie non autorisée est un délit.

Les douze restantes reçoivent en « rétroaction » les signaux de
six des huit sorties ainsi que leurs compléments. Les huit sor-
ties (broches 12 à 19) étant équipées de tampons « 3 états », il
est possible de les placer individuellement en état haute impé-
dance soit de façon permanente, soit en fonction de certains
états d'entrée. À ce moment, six d'entre elles (broches 13 à 18)
peuvent être utilisées comme entrées supplémentaires, ce qui
porte à seize le nombre maximal d'entrées pouvant être exploi-
tées à la fois (avec deux sorties seulement bien sûr).

Figure 3.5.
Schéma interne du
PAL 16L8.

Chacune des huit portes OU pilotant les sorties possède sept entrées, et peut donc combiner sept termes de produit issus de cinquante-six portes ET à trente-deux entrées. Il reste donc huit termes de produit sur les soixante-quatre disponibles (les 0, 8, 16, 24, 32, 40, 48, et 56). Ceux-ci sont affectés à la commande séparée des huit tampons de sortie « 3 états », qui bénéficie ainsi du maximum de souplesse possible.

À ce niveau de notre découverte des PAL, il est temps de noter deux règles fondamentales :

- toute combinaison d'un signal et de son complément dans un terme de produit met celui-ci dans un état bas permanent (cas qui se produit notamment si tous les fusibles d'une ligne sont intacts, en particulier lorsque le PAL n'est pas encore programmé) ;
- si tous les fusibles d'une ligne sont détruits, alors le terme de produit correspondant reste en permanence dans l'état haut.

Pour positionner un tampon de sortie en permanence dans le mode haute impédance, il suffit donc de laisser intacts tous les fusibles de la ligne pilotant son entrée de validation. Par contre, pour transformer ce tampon en simple inverseur de façon permanente, il faut absolument détruire tous les fusibles de cette même ligne de validation.

Le PAL 16R8

La figure 3.6 reproduit le schéma interne du PAL 16R8, avec les mêmes conventions que précédemment.

On remarque la présence de huit entrées seulement, et de huit sorties équipées chacune d'une bascule « D » suivie d'un tampon « 3 états ».

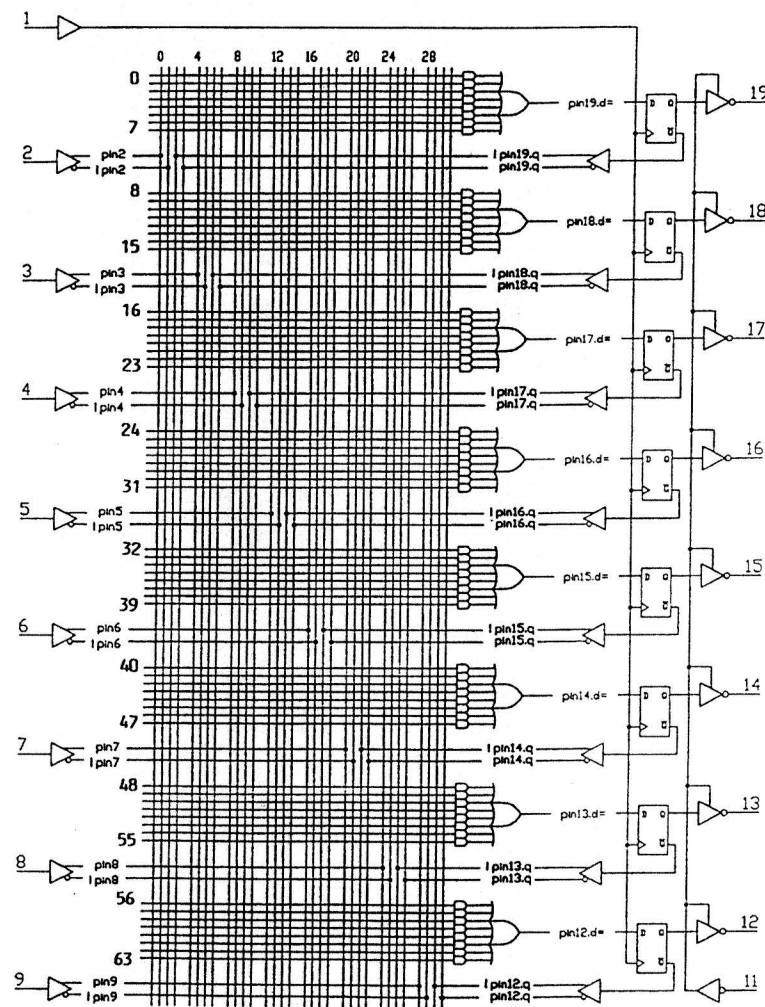
Contrairement au PAL 16L8, les huit tampons sont commandés par une ligne de validation commune rejoignant la broche n°11, tandis que les sorties ne sont pas renvoyées en rétroaction. On ne pourra donc pas les utiliser en tant qu'entrées supplémentaires. Par contre, les états de sortie des huit bascules sont, eux, renvoyés en rétroaction avec leurs compléments, portant bien à trente-deux le nombre des verticales de la matrice programmable.

Sachant qu'il est possible de transformer une bascule D en bascule T, RS, ou même JK en la faisant simplement précéder d'un circuit combinatoire approprié, la configuration du PAL 16R8

permet de réaliser toutes sortes de compteurs ou registres à décalage nécessitant jusqu'à huit bascules.

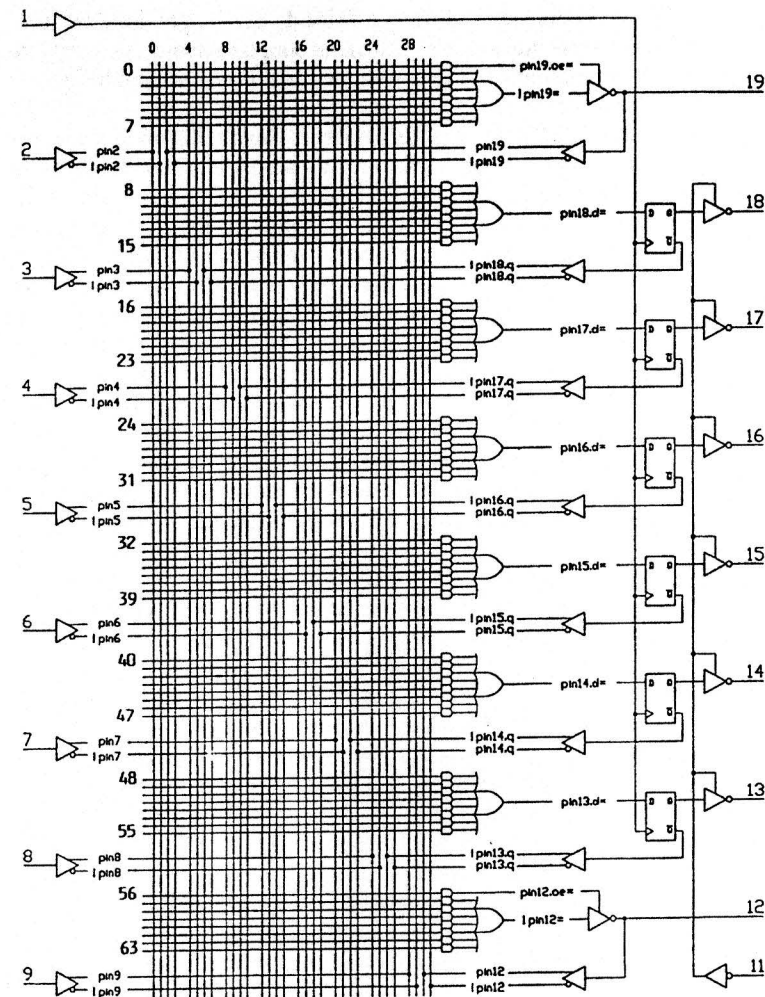
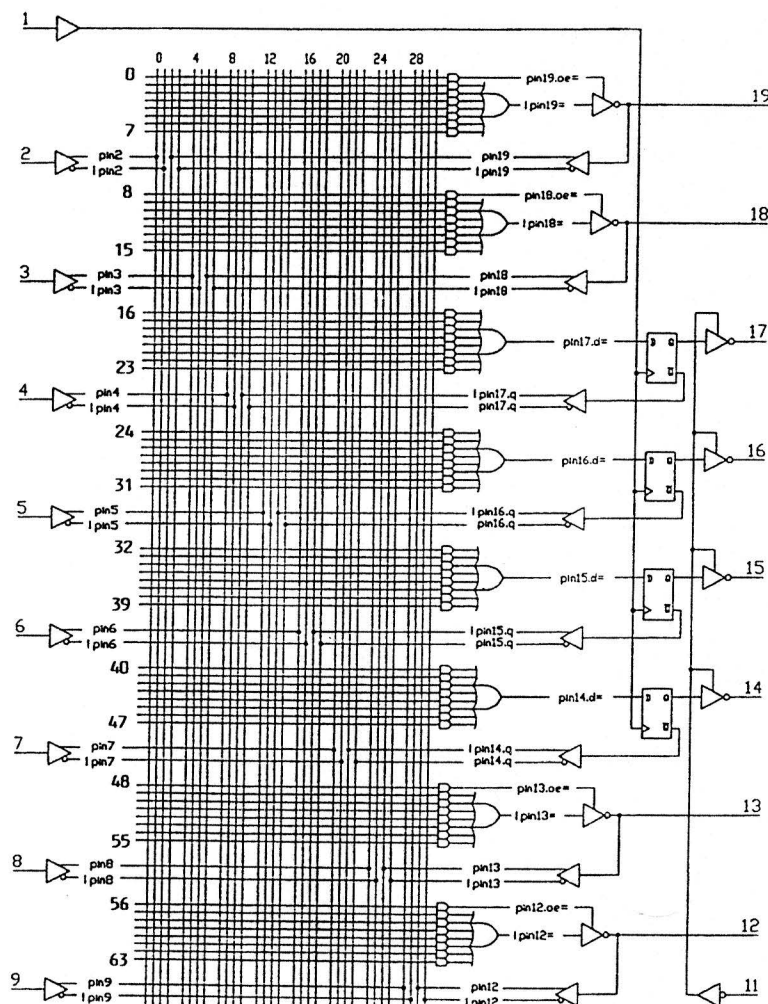
Les entrées d'horloge des huit bascules étant reliées ensemble sur une ligne rejoignant la broche n°1, on devra cependant se limiter surtout à des systèmes séquentiels de nature synchrone.

Figure 3.6.
Schéma interne
du PAL 16R8.



Les PAL 16R4 et 16R6

Figure 3.7.
Schéma interne du
PAL 16R4.



LES PAL UNIVERSELS ET LES GAL

Nous venons déjà de répertorier quatre types différents de PAL, en nous limitant aux plus courants. Il est évident qu'il en existe encore bien d'autres.

Nous avons également expliqué que le relatif manque de souplesse de l'architecture PAL ne peut guère être compensé que par le choix d'un type bien précis pour chaque application. Faut-il donc nécessairement stocker de multiples références ? Pas forcément !

Figure 3.8.
Schéma interne du
PAL 16R6.

Plusieurs fabricants ont eu l'idée de développer des PAL «universels» qui, grâce à des fusibles supplémentaires, peuvent être programmés de façon à simuler différents PAL standards.

Le GAL22V10 est sans doute le plus populaire des réseaux logiques programmables, après les PAL à 20 broches que nous venons d'étudier et qu'il tend de plus en plus à supplanter. Bien que le terme GAL (*Generic Array Logic*) soit une marque déposée de Lattice, il est depuis longtemps utilisé par d'autres fabricants qui en produisent ou qui en ont produit. Certains ont préféré lui inventer un synonyme, par exemple AMD avec sa famille PALCE, ou ICT avec ses PEEL.

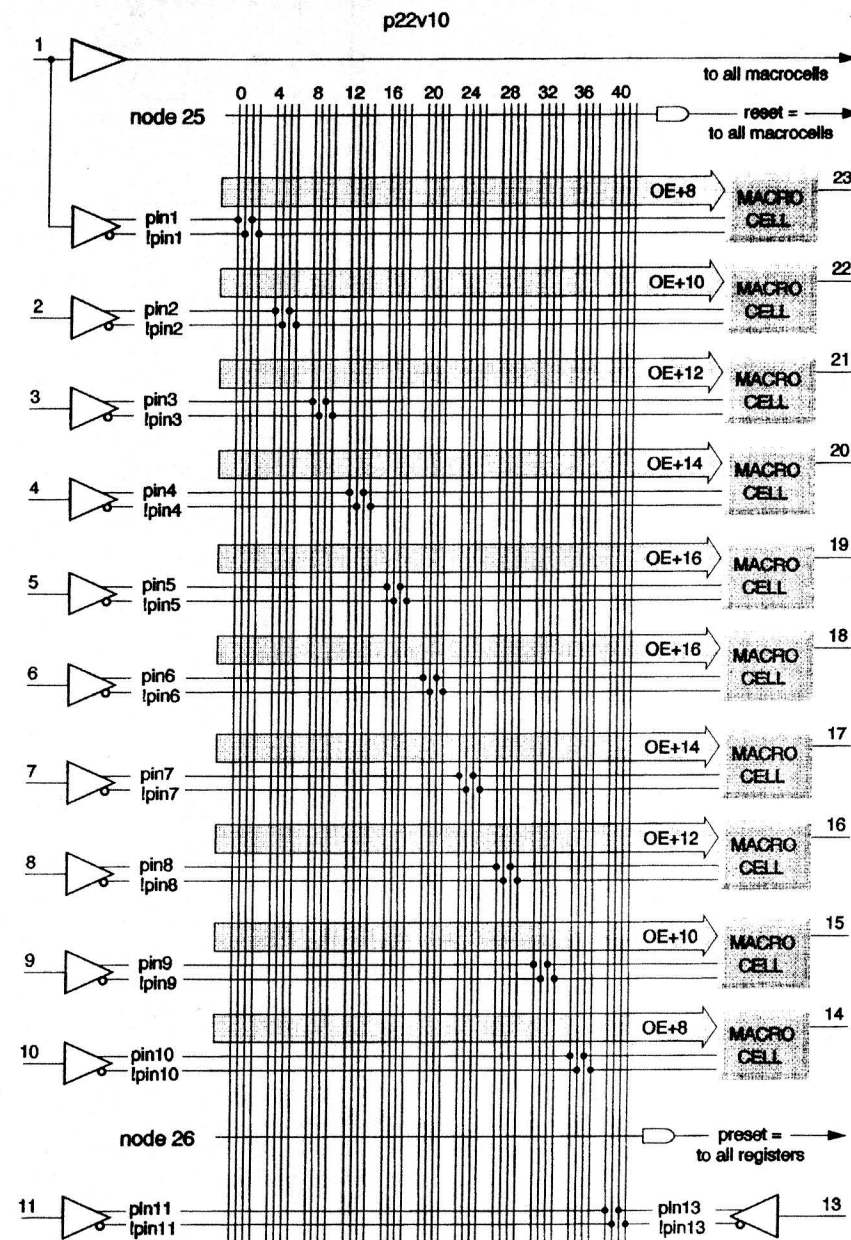
Le principe GAL est en fait une variante de l'architecture PAL dans laquelle les étages de sortie sont eux-mêmes programmables. Ainsi, là où on trouve une bascule ou un tampon dans un PAL comme le 16R6, par exemple, un GAL est muni d'une macrocellule pouvant indifféremment être programmée en bascule ou en tampon, à sortie directe ou complémentée. C'est évidemment beaucoup plus souple, et à raison de dix macrocellules pour 22 entrées possibles, le GAL22V10 est capable de résoudre à lui seul des problèmes logiques déjà conséquents (sa densité atteint l'équivalent de 500 portes).

Le schéma de la figure 3.9 détaille l'organisation de la matrice programmable commune à toutes les macrocellules, tandis que la figure 3.10 montre les quatre configurations possibles pour chaque macrocellule.

Extraits du manuel du compilateur logique Prologic diffusé par Texas Instruments et inclus sur la disquette de ce livre, ces deux documents fournissent les noms des signaux qu'il faudra utiliser pour lui préparer un fichier source à partir d'équations booléennes, de tables de vérité, ou de diagrammes d'état.

La configuration de chaque macrocellule selon l'un ou l'autre des schémas possibles découle automatiquement de l'utilisation des noms de signaux qui lui correspondent (pin23.d=, pin23=, ou !pin23=), et d'une déclaration de polarité dans le cas d'une bascule (pin23= q ; ou !pin23= q ;).

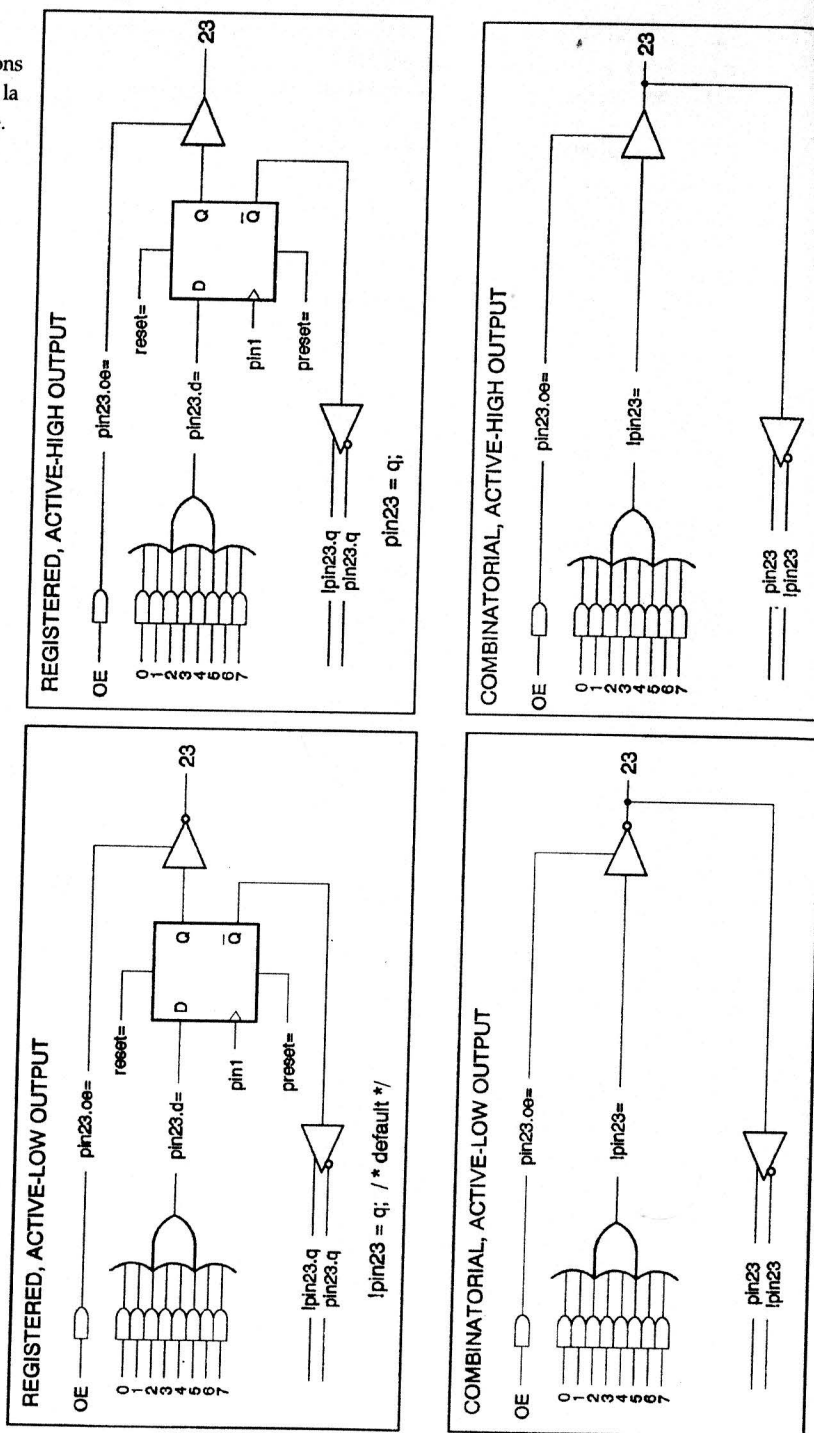
À l'issue de la compilation d'un code-source parfaitement au point, Prologic fournit un fichier indiquant au programmeur les fusibles à détruire ou à laisser intacts dans la matrice d'interconnexion et les macrocellules du GAL.



© DUNOD - La photocopie non autorisée est un délit.

Figure 3.9. Matrice programmable du GAL22V10.

Figure 3.10.
Configurations
possibles de la
macrocellule.



QUE CONTIENNENT LES PAL ET LES GAL ?

Il doit désormais être bien clair que même les PAL de « bas de gamme » que nous avons présentés permettent de réaliser des fonctions logiques relativement évoluées (combinatoires et/ou séquentielles) dans un très petit nombre de boîtiers, souvent même un seul. Pour ce faire, il faut évidemment définir la liste des connexions internes nécessaires, c'est-à-dire écrire le programme voulu puis le transférer dans le composant.

Le développeur habitué à la création de systèmes logiques pourrait évidemment définir la programmation de la matrice d'un PAL en portant des croix sur une photocopie d'un des schémas vierges des figures 3.5 à 3.9.

Souvent, il serait cependant moins fastidieux de construire une table de vérité ou un diagramme d'états, ou simplement d'écrire une série d'équations booléennes.

LES LOGICIELS D'AIDE AU DÉVELOPPEMENT

On ne conçoit plus guère aujourd'hui de développer des applications à base de PAL ou de GAL sans recourir aux logiciels dits « compilateurs logiques », généralement destinés à tourner sur compatibles PC.

Les plus connus sont Palasm (le plus ancien, de chez AMD/MMI), Abel (de Data I/O), Cupl (de Assisted Technology), Plan (de National Semiconductor), et Prologic (de Inlab, distribué en version limitée par Texas Instruments), mais il en apparaît régulièrement de nouveaux, de plus en plus puissants.

En général, les compilateurs capables de supporter les composants de la plupart des marques et même de choisir un type en fonction de l'application coûtent cher : typiquement quinze mille francs et plus.

Par contre, certains fabricants de PAL ou de GAL offrent gracieusement des versions supportant exclusivement leurs propres produits, dans le but à peine voilé d'orienter le développeur vers l'achat de ceux-ci.

Le rôle du compilateur logique consiste à traduire des équations booléennes, des tables de vérité, ou des diagrammes d'état (voire même des schémas logiques) en un « fichier JEDEC », liste codée sous une forme normalisée des fusibles à détruire.

La figure 3.11 illustre un cas particulièrement simple, la réalisation sous Prologic d'une porte NAND à trois entrées avec un PAL 16R4.

Figure 3.11.
Fichier source
NAND3.PLD.

```
include p16r4;
!pin19 = pin2 & pin3 & pin4;
pin19.oe = 1;
test_vectors (
  pin2 pin3 pin4 !pin19;
    0    0    0    L ;
    0    0    1    L ;
    0    1    0    L ;
    0    1    1    L ;
    1    0    0    L ;
    1    0    1    L ;
    1    1    0    L ;
    1    1    1    H ;
)
```

La ligne Include sert à déclarer le type de PAL utilisé, afin que le compilateur puisse utiliser la bonne base de données lors de l'exécution du travail. Suit l'équation, rédigée sous la forme d'une ligne de texte utilisant des signes conventionnels : point d'exclamation pour la complément, « et anglo-saxon » pour le ET logique, etc.

Une ligne indépendante sert à fixer un niveau haut permanent sur la ligne de validation du tampon desservant la broche 19. Notons que le nom de ce signal est tout simplement extrait de la figure 3.7.

Une rubrique Test vectors est enfin prévue. Facultative, elle sert à définir les réactions que l'on attend du composant programmé, indépendamment des équations préalablement fournies. La concordance est évidemment de la seule responsabilité du programmeur !

Au terme de la compilation, on peut exécuter une simulation logique destinée à vérifier que la programmation du composant donnera bien les résultats escomptés, cela sans même disposer encore du moindre échantillon ni d'un quelconque programmeur !

Ce listing doit être saisi à l'aide d'un simple éditeur de texte, et sauvé sur disquette sous la forme d'un fichier portant l'extension « .PLD ». Dans notre cas, ce fichier sera nommé « NAND3.PLD ». Il suffit alors de lancer le compilateur en frappant « LC NAND3 » pour obtenir en quelques instants le fichier JEDEC nommé « NAND3.JED » reproduit à la figure 3.12.

proLogic Compiler
Texas Instruments V1.97
Copyright (C) 1989 INLAB, Inc.

Figure 3.12.
Fichier JEDEC
NAND3.JED.

```
p16r4 revision 89.2.11

*N_csidp16r4
*QF20
*QV8
*QF2048
*F0
*L0000 11111111111111111111111111111111
*L0032 01110111011111111111111111111111
*C07E6
*V1 NOOONNNNNNNNNNNNNNNNNNN
*V2 NOO1NNNNNNNNNNNNNNNNNN
*V3 NO1ONNNNNNNNNNNNNNNNNNN
*V4 NO11NNNNNNNNNNNNNNNNNN
*V5 N1OONNNNNNNNNNNNNNNNNNN
*V6 N1O1NNNNNNNNNNNNNNNNNN
*V7 N11ONNNNNNNNNNNNNNNNNNN
*V8 N111NNNNNNNNNNNNNNNNNN
*V6DOC
```

Il s'agit de la liste codée des fusibles à détruire (1) et à laisser intacts (0), plus éventuellement un jeu de « vecteurs de test » permettant le contrôle du circuit programmé.

Pour procéder à une simulation, il suffirait de frapper « LS NAND3 » et d'attendre les résultats.

Tout bon programmeur de PAL accepte les fichiers JEDEC et est capable de les transférer dans un PAL vierge. Pour ce faire, le fichier contient quelques informations supplémentaires : le type de PAL (ici 16R4), le nombre de pattes (ici 20), le nombre de vecteurs de test (ici 8), et le nombre de fusibles du composant (ici 2 048).

Très proche du matériel, un fichier JEDEC n'est cependant pas facile à interpréter. Prologic fournit donc un fichier supplémentaire nommé « NAND3.LST », qui constitue la documentation en clair.

L'extrait de la figure 3.13 montre que le « plan des fusibles » est représenté d'une façon très détaillée, accompagné par quelques rappels importants, en anglais.

Le fichier JEDEC peut donc servir à programmer directement un PAL 16R4, mais on peut aussi le convertir en vue de la programmation d'un PAL universel ou d'un GAL.

0123 4567 8901 11 1111 1111 2222 2222 2233
0123 4567 8901 2345 6789 0123 4567 8901

0	----	----	----	----	----	----	----
1	X---	X---	X---	----	----	----	----
2	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
3	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
4	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
5	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
6	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
7	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
8	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
9	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
10	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX
11	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX	XXXX

(.)

[illegible]

pin2 3 4 5 6 7 8 9
pin1 18 17 16 15 14 13 11

```

X   : Cell intact      (JEDEC 0)
-   : Cell programmed (JEDEC 1)

X-  : True input term, Complement register term
-X  : Complement input term, True register term
XX  : Any XX pair in a product term yields product term LOW.
--  : No input term (don't care). A product term comprised
      entirely of -- yields product term HIGH.

```

Figure 3.13 Extrait du plan des fusibles (NAND3.LST).

JEDEC PEEL file Translated from: PAL16R4
nand3.JED

Wed 1-2-1980 0:12:34

```

*DF PEEL
*DD 18CV8
*DM ICT
*QP20
*QF2696
*FO
*
N Output Pin 19      *
L0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 *
L0036 1111 0111 0111 0111 1111 1111 1111 1111 1111 *
L0072 0000 0000 0000 0000 0000 0000 0000 0000 0000 *
L0108 0000 0000 0000 0000 0000 0000 0000 0000 0000 *
L0144 0000 0000 0000 0000 0000 0000 0000 0000 0000 *
L0180 0000 0000 0000 0000 0000 0000 0000 0000 0000 *
L0216 0000 0000 0000 0000 0000 0000 0000 0000 0000 *
L0252 0000 0000 0000 0000 0000 0000 0000 0000 0000 *

N Output Pin 18      *
L0288 0000 0000 0000 0000 0000 0000 0000 0000 0000 *
L0324 0000 0000 0000 0000 0000 0000 0000 0000 0000 *

```

La figure 3.14 montre le résultat d'une telle conversion à destination d'un PEEL 18CV8 d'ICT (à 2 696 fusibles). La figure 3.15 montre le résultat de la conversion à destination d'un EP320 d'ALTERA (à 2 916 fusibles).

Figure 3.15.
Conversion JEDEC
pour l'EP320.

proLogic Compiler
Texas Instruments V1.97
Copyright (C) 1989 INLAB, Inc.

EP320
p16r4 revision 89.2.11

```
*N  
PAL2EPLD Version 1.1 - Nov 18, 1986  
EP320 JEDEC from conversion of PAL JEDEC*  
N_csidp16r4  
*QP20  
*QV8  
*QF2916  
*L0  
111010101111111111111111111111111111  
11000000000000000000110000000000000000  
11000000000000000000110000000000000000  
11000000000000000000110000000000000000  
11000000000000000000110000000000000000  
11000000000000000000110000000000000000  
( . . . . . )
```

Figure 3.16.
Reconstitution des
équations.

proLogic Compiler
Texas Instruments V1.97
Copyright (C) 1989 INLAB, Inc.

p16r4 revision 89.2.11

JED2EQN version: 1.0 beta 7 processed: Wed Jan 02 00:39:38 1980
PART: auto

INPUTS:

pin1, pin2, pin3, pin4, pin11

OUTPUTS:

pin14, pin15, pin16, pin17, pin19

NETWORK:

```
pin1 = INP(pin1)
pin2 = INP(pin2)
pin3 = INP(pin3)
pin4 = INP(pin4)
pin11 = INP(pin11)
pin14 = RONF(din14, clk1, GND, GND, oen11)
pin15 = RONF(din15, clk1, GND, GND, oen11)
pin16 = RONF(din16, clk1, GND, GND, oen11)
pin17 = RONF(din17, clk1, GND, GND, oen11)
pin19 = CONF(din19, VCC)
```

EQUATIONS:

```
clk1 = pin1
oen11 = !pin11
!din14 = GND
!din15 = GND
!din16 = GND
!din17 = GND
!din19 = pin2 & pin3 & pin4
END$
```

N'insistons pas davantage sur les profondes différences internes qui existent visiblement entre ces différents composants qui, une fois programmés, deviennent cependant équivalents broche pour broche ! Saisissons par contre l'occasion de découvrir que la syntaxe des fichiers JEDEC est relativement souple, et admet de multiples variantes.

La figure 3.16 illustre pour sa part un traitement logiciel très différent, exécuté à l'aide de l'utilitaire « JED2EQN » fourni par Altera : la reconstitution des équations logiques à partir du fichier JEDEC !

Bien entendu, le but de cette traduction est de produire un fichier-source exploitable par le seul compilateur logique de la marque, mais le résultat de l'opération peut tout de même être consulté dans d'autres buts, par exemple à des fins de contrôle des résultats de la compilation.

Pour en terminer avec les logiciels utilisables, précisons que si un fichier JEDEC peut éventuellement être modifié ou même créé de toutes pièces à l'aide d'un simple éditeur de texte, il existe des utilitaires spécialisés.

La disquette de démonstration des programmeurs Sprint, notamment, comporte un excellent éditeur de fusibles. Les modifications se font sur le « plan des fusibles » en clair, mais les sauvegardes sur disquette sous la forme d'authentiques fichiers JEDEC.

Occupant normalement plusieurs disquettes, cet outil particulièrement utile est reproduit sur le CD-ROM accompagnant notre ouvrage *Logiciels PC pour l'électronique*, paru dans cette même collection.

LES PROGRAMMATEURS

Qu'il soit construit manuellement ou issu d'un compilateur, un fichier JEDEC peut être exploité par n'importe quel programmeur de PAL ou de GAL pour programmer un composant vierge et, généralement, vérifier le résultat de l'opération.

Si tous les PAL de même architecture interne sont compatibles entre eux au niveau du fichier JEDEC, les algorithmes de programmation varient énormément d'une marque à l'autre.

Par algorithme de programmation, il faut entendre le détail des opérations à exécuter pour détruire un fusible donné : quels niveaux de tension appliquer à quelles broches et pendant combien de temps.

On ne programme évidemment pas du tout de la même façon un PAL bipolaire à fusibles métalliques, et un PAL CMOS utilisant une technologie « EPROM ». Mais on ne programme pas non plus de la même façon un PALC 16R8 de Cypress et un TICPAL 16R8 Texas Instruments, tous deux CMOS, sous peine de vives surprises ! Les programmeurs du commerce sont donc fournis avec de gros logiciels pour PC qui, organisés en bases de données, sont capables de configurer le programmeur pour chaque type de PAL ou GAL pouvant être programmé.

Les plus simples supportent une dizaine de références et coûtent quelques milliers de francs, tandis que les plus complets reconnaissent des milliers de références et coûtent des dizaines de milliers de francs. Bien entendu, des mises à jour logicielles périodiques permettent de prendre en compte les composants programmables récemment introduits.

La plupart du temps, ces programmeurs performants peuvent aussi programmer une foule d'autres composants : EPROM, PROM, FPLA, EEPROM, mémoires FLASH, et même microcontrôleurs. Inversement, certains programmeurs d'EPROM peuvent recevoir des adaptateurs pour certains PAL, GAL, ou EPLD. Ce genre d'équipement est évidemment presque indispensable à qui souhaite tirer le maximum de la technique des composants programmables, c'est-à-dire utiliser pour chaque projet le ou les PLD répondant de façon optimale au problème posé. Mais un compilateur logique de très haut de gamme sera son complément quasiment indissociable...

Un tel investissement est à l'évidence hors de portée de l'utilisateur occasionnel, ou du développeur souhaitant simplement commencer à « se mettre à la logique programmable » tranquillement et en toute liberté.

On peut songer à l'approche suivante, particulièrement économique :

- choisir quelques références de composants courants, faciles à approvisionner et peu coûteux. S'y tenir même si leur efficacité ne semble pas optimale pour tous les projets ;
- acquérir quelques pièces en technologie effaçable afin de pouvoir manipuler par essais successifs ;
- se procurer un compilateur logique gratuit, ou commencer par développer à la main ;
- construire par soi-même un programmeur simplifié.

Il est cependant indispensable de disposer de l'algorithme de programmation détaillé des composants que ledit programmeur devra supporter, ce qui limite quelque peu les possibilités de choix.

LES « GRANDS » EPLD

Il ne fait aucun doute qu'au-delà des PAL et GAL, l'avenir de la logique programmable se situe du côté des composants rassemblant l'équivalent de quelques milliers de portes dans un boîtier CMS à 44 broches ou davantage.

Jusqu'à présent, aborder ce domaine supposait un investissement lourd en matériel, en logiciel, et... en apprentissage.

Les techniques modernes de programmation « en circuit » (isp) ou « JTAG » révolutionnent littéralement ce marché en mettant véritablement à la portée de tous la création de circuits LSI spécifiques.

Il n'est sans doute pas inutile de rappeler brièvement le vocabulaire employé pour définir le degré d'intégration des composants logiques.

On parle de SSI (*Small Scale Integration*) dans le cas de circuits intégrés regroupant quelques portes simples, ou tout au plus une ou deux bascules séparées. Dans tous les cas, la densité d'intégration ne dépasse guère l'équivalent d'une bonne dizaine de portes.

La catégorie MSI (*Medium Scale Integration*) rassemble les fonctions de base un peu plus évoluées des séries « 74 » ou « 4000 » : décodeurs, multiplexeurs, démultiplexeurs, et assemblages pas trop complexes de bascules comme les compteurs, les registres, ou les verrous (l'équivalent de quelques dizaines de portes, au plus une petite centaine).

On commence à aborder la LSI (*Large Scale Integration*) dès qu'un seul boîtier abrite une fonction complexe spécifique (synthétiseur de fréquence, décodeur de codes à barres, UART, CI de télécommande, etc.) On atteint alors des densités correspondant à l'équivalent de quelques centaines à quelques milliers de portes.

Enfin, on ne parle guère de VLSI (*Very Large Scale Integration*) qu'à partir de plusieurs milliers de portes, et jusqu'à des dizaines de milliers : l'équivalent en complexité d'un microprocesseur, certains VLSI étant d'ailleurs tout simplement des microcontrôleurs dont la ROM (masquée) est programmée d'une façon spécifique.

Les composants logiques programmables offrent des alternatives, la plupart du temps fort élégantes, à l'assemblage de composants standards de ces différentes catégories. Un PAL ou un GAL remplacera couramment jusqu'à une dizaine de boîtiers SSI ou bien deux ou trois boîtiers MSI, parfois même davantage. Un GAL22V10 équivaut par exemple à environ 500 portes, mais il est bien difficile toutefois de tendre vers une utilisation à 100 % de ces ressources.

Il faudrait donc en associer plusieurs pour arriver, sans doute assez laborieusement, à l'équivalent des LSI les plus simples. Aussi les fabricants de semi-conducteurs ont-ils imaginé des structures programmables beaucoup plus vastes, adaptées aux projets logiques les plus ambitieux.

Les familles MACH d'AMD ou pLSI de Lattice sont des exemples parfaitement représentatifs, dont certaines références bénéficient en plus du concept de programmation « en circuit », c'est-à-dire quasiment sans programmeur...

Grâce à des logiciels de développement exceptionnellement faciles à utiliser et parfois très peu coûteux, la création de circuits LSI personnalisés est maintenant presque plus simple que le développement d'applications à base de PAL !

LES pLSI ET ispLSI DE LATTICE

La famille de composants programmables pLSI a réussi à propulser Lattice (l'inventeur du premier GAL) en bonne place parmi les fabricants de réseaux logiques programmables (PLD) à haute densité. Cette gamme de composants est en effet plutôt bien située face à une concurrence féroce, sur un marché en pleine effervescence.

Il faut savoir que la puissance d'un circuit LSI programmable est liée tout à la fois aux fonctions logiques qu'il contient, au nombre de broches d'entrée-sortie qu'il offre, et aux possibilités d'interconnexion (routage) de l'ensemble de ces ressources.

Avec ses familles baptisées p/isp LSI1000, 2000, et 3000, Lattice couvre le domaine s'étendant de 1 000 à 14 000 portes, dans des boîtiers comptant entre 44 et 208 broches, et ce n'est sans doute qu'un début.

D'un point de vue purement logique, les trois familles sont basées sur les notions de GLB (*Generic Logic Block*), de GRP (*Global Routing Pool*), d'ORP (*Output Routing Pool*), et d'I/O cell.

Le GLB rappelle, par son architecture et ses possibilités de programmation, la structure des GAL. La figure 3.17 montre en effet qu'il s'agit d'un bloc logique muni de 18 entrées et de 4 sorties, d'une matrice programmable de 20 portes ET (termes de produit), et de 4 macrocellules programmables individuellement en mode combinatoire (ET, OU, OU exclusif) ou en mode séquentiel (D, T, J/K, synchrone ou asynchrone).

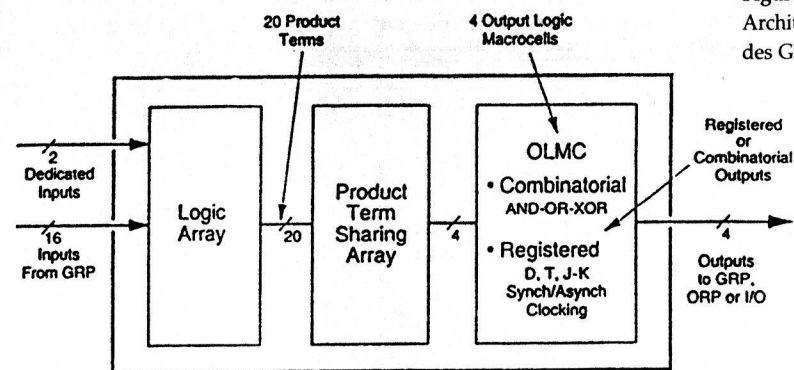


Figure 3.17.
Architecture
des GLB.

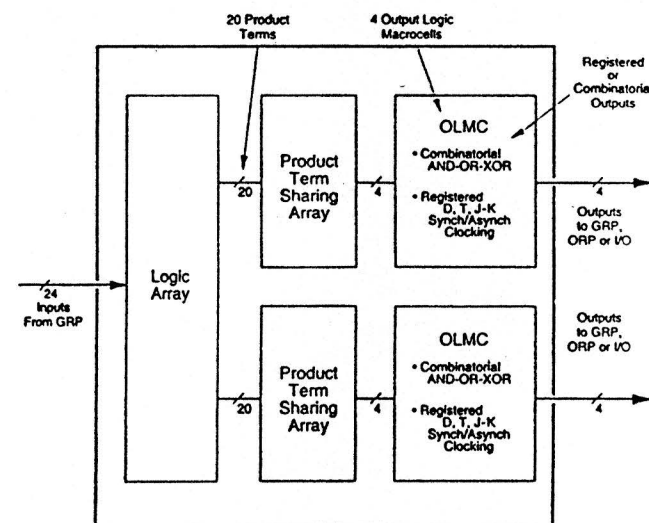


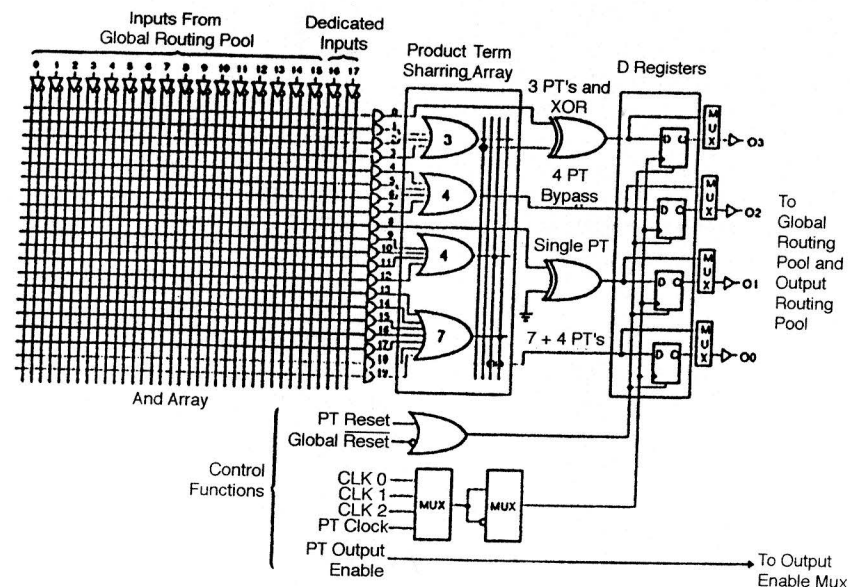
Figure 3.18.
Un GLB « double ».

Notons que les composants de la famille 3000 bénéficient de GLB « doubles » (figure 3.18) à 24 entrées, 40 termes de produit, et 8 macrocellules.

Un GLB « simple » (celui des séries 1000 et 2000) est capable de reconstituer, par simple programmation, près de 90 % des fonctions logiques usuelles à 4 bits. Et pour travailler sur des octets, on en associe tout simplement deux !

Figure 3.19.

Quelques exemples de mise en œuvre des macrocellules.



S'il fallait risquer une comparaison avec le monde des GAL, on pourrait considérer un GLB simple comme un « 18V4 » imaginaire, deux GLB simples ou un GLB double se rapprochant d'assez près d'un GAL22V10.

Cette parenté des « grands » EPLD avec les PAL et les GAL est tout aussi évidente dans le cas du MACH 130, composant de marque AMD auquel une application très populaire a donné ses lettres de noblesse...

La figure 3.20, qui détaille la structure de son bloc logique de base, permet de comprendre comment un MACH 130 peut être considéré comme la réunion « intelligente » de quatre « 26V16 » imaginaires.

Son boîtier PLCC à 84 connexions n'en demeure pas moins compatible broche pour broche avec les MACH 230 (équivalent à huit 26V16), MACH 231 (huit 32V16), et MACH 435 (huit 33V16).

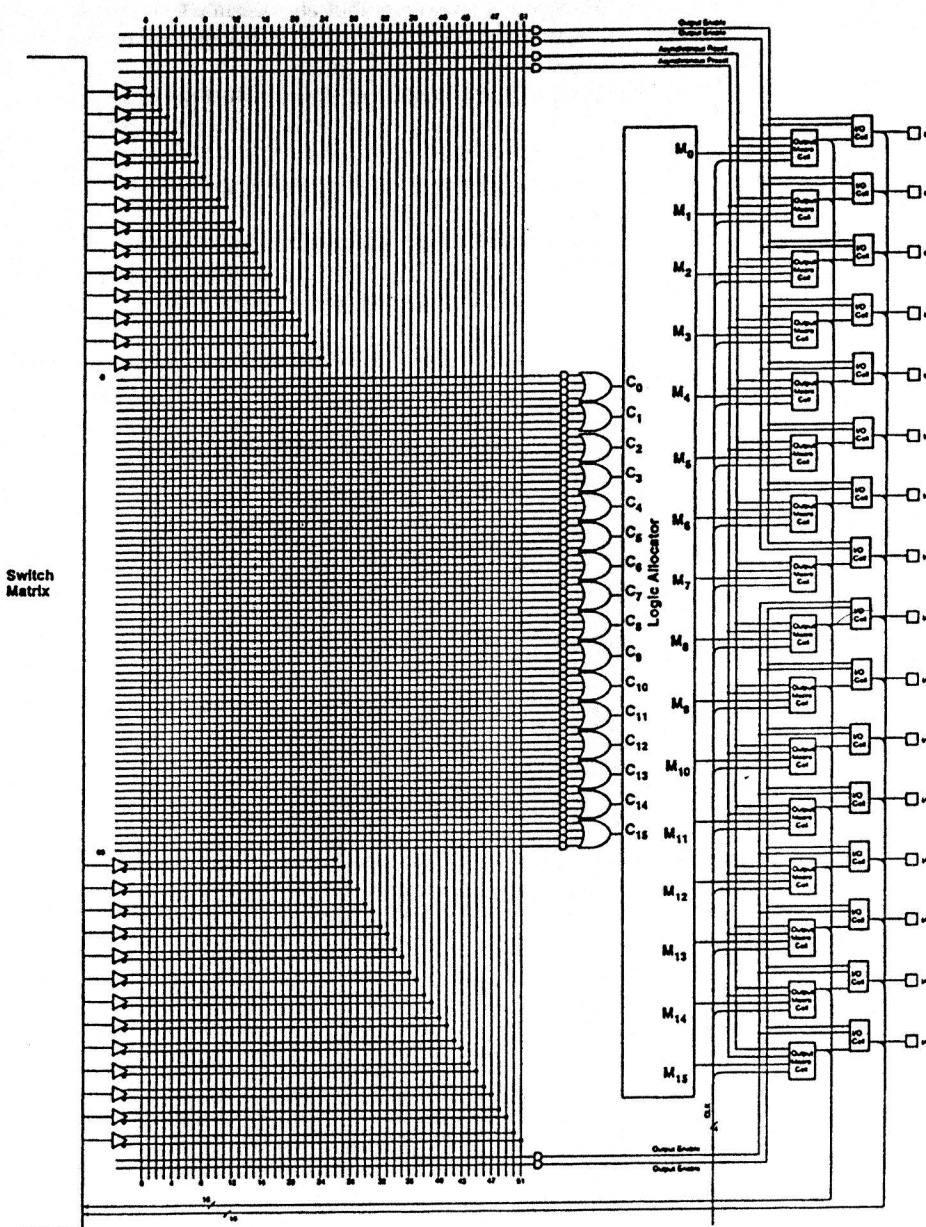


Figure 3.20. Bloc logique de base du nMACH 130.

Présenté en boîtier PQFP à 144 broches, le MACH 355 est pour sa part équivalent à six 33V16, tout en présentant l'immense avantage d'être programmable « en circuit » selon le protocole JTAG, autrement dit... sans programmeur.

Moyennant quelques adaptations, il serait ainsi parfaitement envisageable de remplacer, dans certaines applications, le MACH 130 par le MACH 355, auquel ses possibilités de télé-chargement ouvrent naturellement des perspectives encore plus vastes.

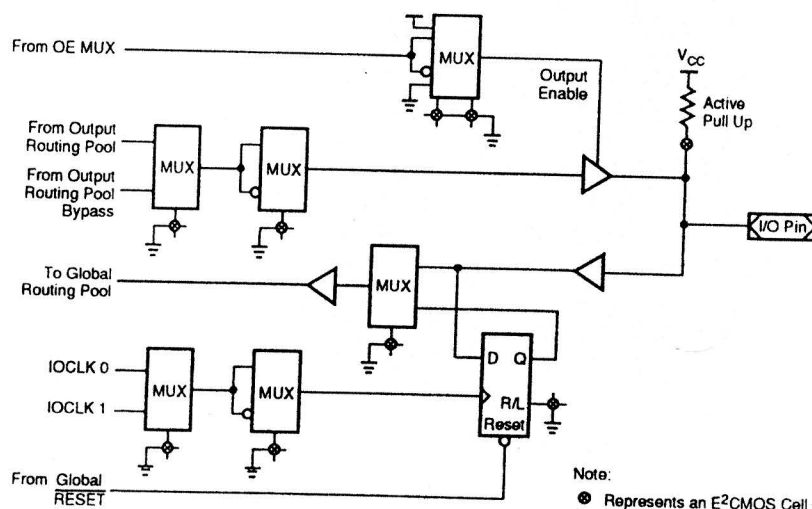
Chez Lattice, le plus simple LSI de la série 1000 (l'ispLSI 1016) contient déjà 16 GLB, tandis que les GLB ne constituent jamais qu'une des quatre « couches » logiques programmables disponibles.

En amont des GLB se trouve en effet le GRP qui, comme son nom l'indique, est chargé d'assurer des interconnexions librement définies (mais à temps de propagation constant) entre toutes les ressources logiques du composant, et aussi avec ses broches d'entrée.

En aval des GLB, on rencontre cette fois les ORP, chargées de diriger les sorties des macrocellules vers des broches librement choisies, cela à raison d'un ORP pour quatre ou huit GLB. Mais certaines sorties de chaque GLB peuvent aussi rejoindre directement les cellules d'entrée-sortie correspondantes sans passer par l'ORP. On y gagne en rapidité !

Couche programmable à part entière, l'ensemble des cellules d'entrée-sortie fonctionne selon le principe détaillé à la figure 3.21.

Figure 3.21.
Structure
d'une cellule
d'entrée-sortie.



Mais il convient de noter qu'en plus de ces broches d'entrée-sortie universelles, chaque référence de LSI possède quelques broches spécialisées : entrées dédiées (inutilisables en tant que sorties), entrées d'horloge, et éventuellement entrées de pilotage global des OE. Leur nombre et leur rôle varie légèrement d'une référence à l'autre.

Tous les composants de la gamme p/isp LSI contiennent un nombre entier de megablocks, assemblages de GLB, de cellules d'entrée-sortie, d'entrées dédiées et d'ORP, avec un terme de produit « OE » commun. Leur contenu exact diffère quelque peu d'une famille à l'autre.

La famille 1000 est riche de quatre références : p/isp LSI1016, 1024, 1032 et 1048, munies respectivement de 2, 3, 4, et 6 megablocks (l'équivalent de 2 000, 4 000, 6 000, et 8 000 portes).

La famille 2000 commence avec un seul megablock (p/isp LSI2032 à 1 000 portes), mais la famille 3000 en prévoit jusqu'à dix (p/isp LSI3320 à 14 000 portes).

Comme il ne fait aucun doute que les produits les plus populaires, toutes marques confondues, sont pour l'instant ceux à 44 broches (et autant que possible programmables « en circuit » !), il est tout indiqué de s'intéresser en priorité aux ispLSI2032 et 1016. Ce sont précisément ces deux références qui sont supportées par le logiciel du starter kit qui réunit, pour un prix fort modique : un cordon de programmation, quelques échantillons de composants compatibles (dont un ispGAL22V10...), les logiciels nécessaires et de la documentation.

Le tableau de la figure 3.22 compare les ressources de ces deux composants avec, pour mémoire, celles des autres membres de la famille 1000.

À l'usage, on se rendra vite compte qu'avec 1 000 portes, un ispLSI2032 est incomparablement plus puissant qu'un GAL22V10 qui en contient tout de même au moins l'équivalent de 500. La différence provient essentiellement d'une souplesse de programmation sans commune mesure.

Le logiciel de développement que contient l'isp Starter Kit est en tout point identique, y compris par le volume de sa documentation, au produit « PDS » supportant toute la gamme p/isp LSI, à ceci près qu'il est limité aux références 2032 et 1016.

Moyennant un investissement très supportable, on pourra ainsi développer des applications dépassant de très loin les possibilités des GAL et à plus forte raison des PAL, tout en bénéficiant

de toute la puissance d'un outil professionnel étonnamment convivial. L'une des raisons de cette convivialité est le fonctionnement sous Windows, mais le plus surprenant pour un outil de ce prix est la présence d'une bibliothèque de macros extrêmement fournie.

Figure 3.22.
Principales
références de la
famille ispLSI.

	ispLSI2032	ispLSI1016	ispLSI1024	ispLSI1032	ispLSI1048
Macrocellules	32	64	96	128	192
GLB	8	16	24	32	48
Megablocks	1	2	3	4	6
Équivalent portes	1000	2000	4000	6000	8000
Broches d'E/S	32	32	48	64	96
Broches d'entrée	2	4	6	8	10
Nb broches boîtier	44 (PLCC) 44 (TQFP)	44 (PLCC) 44 (TQFP) 44 (JLCC)	68 (PLCC) 68 (JLCC)	84 (PLCC) 84 (CPGA) 100 (TQFP)	120 (PQFP) 128 (PQFP)
Registres	32	96	144	192	288
F max.	135 MHz	110 MHz	90 MHz	90 MHz	80 MHz

Pratiquement toutes les fonctions logiques disponibles en SSI et en MSI peuvent être directement prélevées dans un épais data book (fourni) dont la figure 3.23 reproduit une page.

Le décodeur 7 segments « BIN27 » (qui décode même l'hexa-décimal, s'il vous plaît !) peut ainsi être incorporé dans un coin du composant en cliquant tout simplement sur son nom dans une liste. Par « dans un coin », il faut entendre ici « à cheval sur deux GLB ». En effet, le décodeur possède sept sorties et ne peut évidemment pas tenir dans un seul GLB qui en dispose seulement de quatre. Mais beaucoup d'autres macros tiennent à l'aise dans un seul GLB, dont on rappelle que l'ispLSI2032 en possède déjà huit.

Tout développement autour d'un ispLSI commencera donc sur le papier, par un « découpage » du schéma synoptique en plusieurs fonctions tenant chacune dans un ou plusieurs GLB, ce qui permettra de savoir d'emblée si son intégration est possible dans le composant prévu. Cela étant fait, on pourra commencer à travailler avec le logiciel PDS, qui affiche sur l'écran une représentation stylisée du schéma du composant utilisé.

BIN27

Decoders

Type: Soft Macro

Logic Function

Name	Function
BIN27	Binary to seven segment decoder with enable

Truth Table

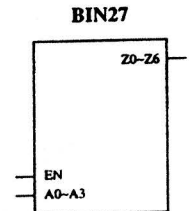
Input					Output							Character
EN	A3	A2	A1	A0	Z0	Z1	Z2	Z3	Z4	Z5	Z6	
1	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
1	0	0	1	0	1	1	0	1	1	0	1	2
1	0	0	1	1	1	1	1	1	0	0	1	3
1	0	1	0	0	0	1	1	0	0	1	1	4
1	0	1	0	1	1	0	1	1	0	1	1	5
1	0	1	1	0	1	0	1	1	1	1	1	6
1	0	1	1	1	1	1	1	0	0	0	0	7
1	1	0	0	0	1	1	1	1	1	1	1	8
1	1	0	0	1	1	1	1	1	0	1	1	9
1	1	0	1	0	1	1	1	0	1	1	1	A
1	1	0	1	1	0	0	1	1	1	1	1	B
1	1	1	0	0	0	0	0	1	1	0	1	C
1	1	1	0	1	0	1	1	1	1	0	1	D
1	1	1	1	0	1	0	0	1	1	1	1	E
1	1	1	1	1	1	0	0	0	1	1	1	F
0	x	x	x	x	0	0	0	0	0	0	0	Off

Equation Entry

BIN27 ([Z0..Z6],[A0..A3],EN)
BIN27_1([Z0..Z2],[A0..A3],EN)
BIN27_2([Z3..Z6],[A0..A3],EN)

Le gros du travail consiste à « remplir » les GLB et les cellules d'entrée-sortie nécessaires avec des équations booléennes cohérentes entre elles. En effet, il faut bien que le signal entrant dans un GLB, par exemple, porte le même nom que celui qui sort d'une cellule d'entrée-sortie ou d'un autre GLB. Inversement, un signal sortant d'un GLB devra correspondre à un signal de même nom entrant dans une cellule d'entrée-sortie ou dans un autre GLB.

Logic Symbol



Segment Map

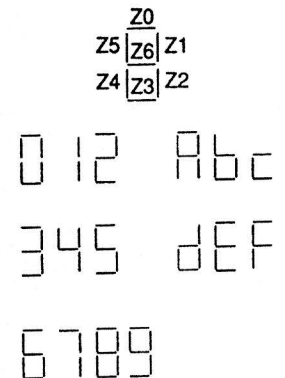


Figure 3.23.
Data-sheet du
décodeur BIN27.

Pour chaque GLB, on devra ainsi renommer manuellement les signaux entrant dans la composition des équations de la macro utilisée, ou écrire directement ses propres équations à partir des noms des signaux réellement utilisés et des opérateurs logiques reconnus par PDS. Cela demande évidemment un certain entraînement...

On mesure là encore tout l'intérêt qu'il y a à bien préparer, sur le papier, le découpage du schéma en blocs fonctionnels dont les signaux d'entrée et de sortie seront parfaitement identifiés.

Comme une erreur est vite commise dans ce genre de processus, de puissantes fonctions de vérification sont incorporées dans PDS. On s'en servira tout d'abord sur chaque GLB (préablement baptisé d'un nom facilitant son identification ultérieure), sur chaque cellule d'entrée-sortie, puis sur le schéma complet. Une fois tout cela bien au point, on pourra déclencher l'opération dite de routage. Il s'agit de faire déterminer par le logiciel les chemins que devront suivre les signaux à l'intérieur du GRP et des ORP.

À cette occasion, on pourra choisir la broche affectée à chaque signal entrant ou sortant, opération qui risque de simplifier singulièrement l'étude du circuit imprimé destiné à accueillir tôt ou tard le composant.

À part quelques broches dont l'affectation est fixe, le concepteur n'a que l'embarras du choix pour faire aboutir tel ou tel signal presque n'importe où sur les quatre côtés du boîtier PLCC.

Au terme de tout ce processus, on obtient un listing en format dit «LDF», un fichier-texte qui pourrait tout aussi bien être écrit directement par un programmeur ayant au préalable appris la syntaxe de ce langage spécifique. C'est ce qui se fait couramment en matière de PAL, avec des compilateurs logiques comme Prologic. Mais dans le cas présent, comment ne pas préférer assembler quelques macros choisies « sur catalogue », en cliquant simplement avec une souris ?

À ce stade, il ne reste plus qu'à demander au logiciel de compiler ce « code-source » en un fichier JEDEC, compatible avec n'importe quel programmeur du commerce supportant les pLSI ou les ispLSI.

Tout l'intérêt des versions « isp » de ces composants réside toutefois dans la possibilité de les programmer « en circuit », par téléchargement à travers un simple bus à quatre fils, autrement dit sans programmeur coûteux !

En phase de production, cela se fera la plupart du temps par un branchement direct sur la carte définitive, parfois même à l'occasion d'un test automatique JTAG.

Mais en cas de nécessité de reprogrammations (même fréquentes) chez l'utilisateur final (fixe ou mobile), on pourra tout aussi bien confier l'opération à un processeur hôte, par envoi de disquette ou par l'entremise d'un modem, téléphonique ou hertzien.

En fait, les ispLSI permettent de bénéficier tout à la fois de la souplesse des réseaux programmables à RAM (qui perdent, eux, leur contenu lors de chaque mise hors tension), et de la non-volatilité de la technologie EECMOS : le meilleur des deux mondes, en quelque sorte !

1	Introduction aux composants programmables	9
2	Les mémoires programmables	13
3	Les réseaux logiques programmables	57

4 LES MICRO- CONTRÔLEURS

Les systèmes à microprocesseur	96
Les microcontrôleurs	98
Les microcontrôleurs EPROM et OTP	99
Les logiciels et les systèmes d'aide au développement	100
Les programmeurs pour microcontrôleurs	105
Les microcontrôleurs PIC	106

5	Réalisation de programmeurs	111
6	La disquette du livre	155

Le microprocesseur est véritablement le composant à tout faire de l'électronique contemporaine. On le trouve en effet aussi bien au cœur des ordinateurs que dans les programmeurs de machines à laver, les moteurs de voiture, ou les commandes d'ascenseurs.

Le microprocesseur est aussi le meilleur exemple possible de logique programmable. Un tel composant reste parfaitement inutilisable tant qu'on ne lui fournit pas un programme ou logiciel, liste détaillée d'instructions à exécuter. Bien plus que le schéma de la partie matérielle, c'est essentiellement ce programme, enregistré dans une mémoire non volatile, qui détermine le comportement du système : un simple changement de programme peut transformer complètement un montage à microprocesseur !

LES SYSTÈMES À MICROPROCESSEUR

La figure 4.1 montre l'architecture classique du système à microprocesseur le plus général. Le microprocesseur proprement dit ou unité centrale est entouré de plusieurs autres circuits annexes, indispensables à son fonctionnement : une alimentation, bien sûr, mais aussi un générateur de fréquence d'horloge, des mémoires, et des périphériques d'entrée-sortie.

Le signal d'horloge, dont la fréquence est souvent fixée par un quartz, sert à cadencer le fonctionnement de l'ensemble du système. C'est lui qui détermine notamment la durée d'exécution des instructions. La mémoire morte (ROM, EPROM, EEPROM ou même FLASH) contient le programme à exécuter, chacune de ses instructions se présentant sous la forme d'un ou plusieurs mots (de 8 bits ou davantage) consécutifs. La mémoire vive (RAM) sert au microprocesseur pour stocker temporairement des informations telles que des résultats intermédiaires de calculs complexes. On la surnomme souvent le bloc-notes.

Les périphériques d'entrée-sortie, pour leur part, prennent en charge les communications avec le monde extérieur : capteurs, relais, moteurs, voyants, mais aussi claviers et écrans s'il y a lieu. En effet, à quoi pourrait donc bien servir un montage incapable de recevoir des informations ou d'en émettre ? Un microprocesseur est donc un circuit intégré qui doit être relié à tous ses composants annexes par de multiples connexions.

En premier lieu un bus de données par lequel transitent les octets échangés avec les mémoires et les périphériques : huit

lignes pour les microprocesseurs 8 bits, 16 ou 32 pour les plus puissants. Ensuite, un bus de commandes précisant si le mot présent sur le bus de données émane de l'unité centrale ou lui est destiné, et s'il concerne les mémoires ou les périphériques : encore quelques lignes, typiquement quatre ou cinq.

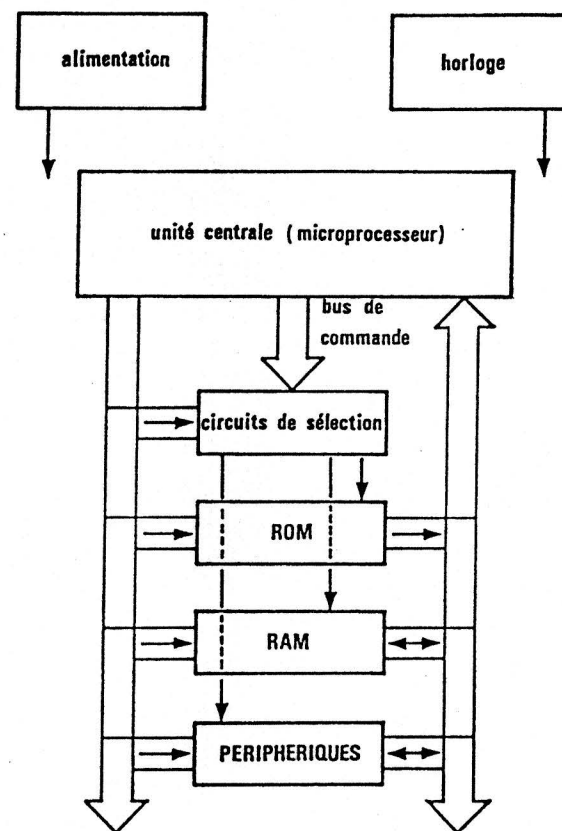


Figure 4.1.
Architecture type
d'un système à
microprocesseur.

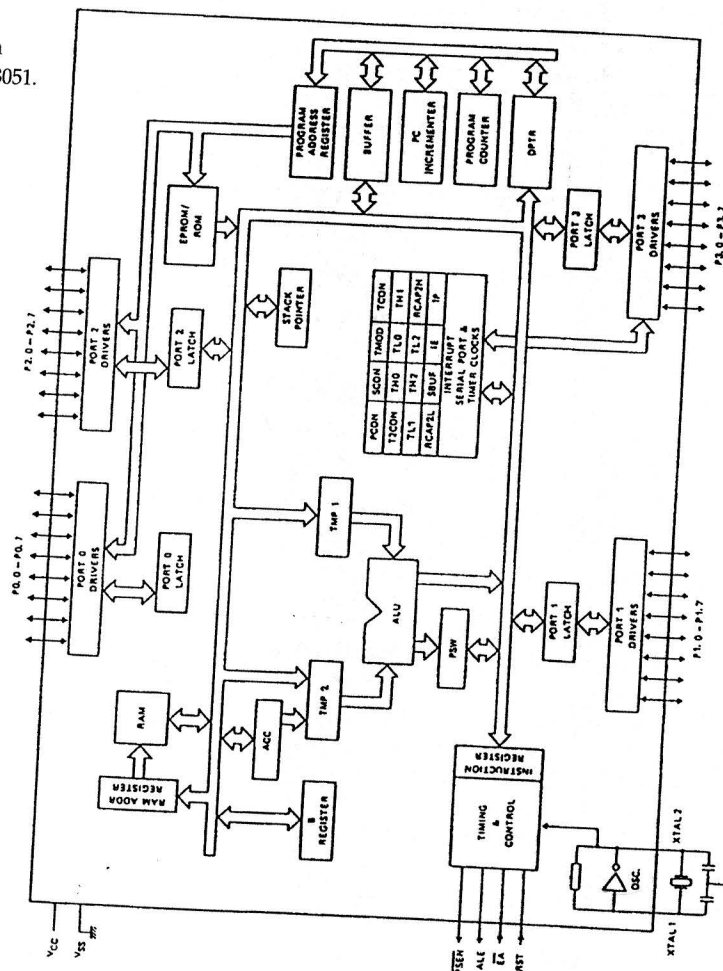
© DUNOD - La photocopie non autorisée est un délit.

Enfin, un bus d'adresses servant à désigner sans équivoque la case mémoire ou le port d'entrée-sortie devant communiquer avec l'unité centrale : seize lignes ou davantage. Sachant que chaque boîtier de mémoire comporte en général 24 ou 28 broches et chaque circuit d'entrée-sortie une vingtaine, on imagine la complexité de l'interconnexion de l'ensemble...

LES MICROCONTRÔLEURS

Un microcontrôleur ou monochip est un circuit intégré rassemblant dans un seul et même boîtier un microprocesseur, de la mémoire, et des circuits d'entrée-sortie. De plus, l'architecture matérielle du composant et son jeu d'instructions sont en général optimisés pour des applications orientées vers le domaine des automatismes.

Figure 4.2.
Architecture d'un
microcontrôleur 8051.



On y gagne une simplification considérable du câblage externe et une miniaturisation sensible du produit fini, le tout pour une dépense souvent inférieure.

Le nombre de broches du boîtier dépend essentiellement de la quantité de lignes d'entrée-sortie nécessaires : un boîtier classique à 40 broches peut facilement accueillir quatre groupes de huit lignes d'E/S, tandis que certains microcontrôleurs très récents arrivent à tenir dans un simple boîtier à 8 broches (PIC12CXXX).

À titre d'exemple, la figure 4.2 reproduit le synoptique interne d'un microcontrôleur appartenant à la famille très courante des 8051. On constate qu'il s'agit bel et bien d'un système à microprocesseur très complet, possédant même des compteurs-temps et un port de communication série.

LES MICROCONTRÔLEURS EPROM ET OTP

Les microcontrôleurs munis d'une mémoire ROM interne «masquée» ne conviennent guère qu'à des fabrications en grande série (électroménager, automobile, etc.).

Le microcontrôleur idéal pour le développement de prototypes est évidemment le modèle à UVPROM incorporée. Il peut être programmé, essayé, effacé, et reprogrammé à volonté aussi facilement qu'une simple EPROM, dont il emprunte le boîtier céramique à fenêtre de quartz.

Et n'oublions pas les versions OTP qui, présentées en boîtier plastique économique, ne peuvent être effacées et sont donc à réserver à la production en petite ou moyenne série de produits parfaitement au point.

Le microcontrôleur EPROM est le composant programmable par excellence : la simple écriture d'un logiciel approprié permet de transformer une pièce standard en un composant spécifique capable de prendre en charge des fonctions qui peuvent être extrêmement complexes. Contrairement aux simples EPROM, les microcontrôleurs peuvent généralement être protégés contre la lecture abusive du contenu de leur mémoire : le logiciel qu'ils renferment peut donc être mis à l'abri des regards indiscrets !

LES LOGICIELS ET LES SYSTÈMES D'AIDE AU DÉVELOPPEMENT

Écrire un programme pour microcontrôleur directement en langage machine est un travail particulièrement fastidieux. Jusqu'à quelques dizaines ou, au plus, quelques centaines d'instructions, on peut à la rigueur opérer à la main, en s'aidant des tables de codes opératoires qui figurent dans les notices techniques des composants. Mais toute correction d'erreur ou modification entraîne la plupart du temps de profonds remaniements du code exécutable. Heureusement, ce genre de travail se prête particulièrement bien à un traitement informatique. Tout micro-ordinateur peut théoriquement être programmé pour assembler du code machine destiné à n'importe quel microprocesseur ou microcontrôleur.

Les logiciels capables d'accomplir cette tâche se nomment assembleurs. On leur fournit un programme source écrit à l'aide de mnémoniques (mots anglais décrivant les instructions du langage assembleur), et ils en déduisent un programme exécutable composé d'octets pouvant être chargés en EPROM.

Dans certains cas, une étape intermédiaire est nécessaire sous la forme d'un ou plusieurs programmes objet qui doivent alors être repris par un *linker* ou éditeur de liens. Inversement, des logiciels connus sous le nom de désassembleurs sont capables de reconstituer une liste de mnémoniques à partir du code exécutable qui, lui, reste pratiquement incompréhensible à la lecture. Une aubaine pour les pirates en tous genres ! Les choses se compliquent dès lors qu'il s'agit de tester le programme : nous sommes très loin du BASIC où un simple *Run* permet un essai immédiat sans risque de « plantage » en cas d'erreur.

Pour essayer réellement un programme de microcontrôleur, il faut en principe le « brûler » dans l'EPROM, le « faire tourner » pour noter les éventuelles anomalies, puis tout reprendre à zéro : modification du code source, réassemblage, effacement puis reprogrammation de l'EPROM.

Diverses techniques permettent fort heureusement de limiter ces manipulations tout en gagnant un temps précieux.

D'abord, la plupart des logiciels assembleurs signalent les erreurs de programmation les plus grossières, permettant leur correction à un stade précoce. Dans le cas où le micro-ordinateur utilisé pour l'assemblage est équipé d'un microprocesseur

de la même famille que le microcontrôleur cible, il est souvent possible de lui faire essayer le programme sous le contrôle d'un logiciel *debugger* ou débogueur permettant de repérer les erreurs et d'éviter le « plantage » pur et simple de la machine.

La plupart du temps, hélas, le microprocesseur de l'ordinateur est totalement différent du microcontrôleur à programmer : le cas habituel est un compatible PC équipé d'un 386 ou d'un 486, tandis que la cible est un 8051, un 68HC11, ou un PIC. Il faut alors recourir à un logiciel dit simulateur, qui reconstitue les réactions du microcontrôleur sur un micro-ordinateur muni d'un processeur quelconque.

À côté de cette approche purement logicielle existe aussi une solution matérielle : l'émulateur.

Un émulateur est un accessoire qui se branche sur le micro-ordinateur servant à l'assemblage, et qui possède un câble muni d'un connecteur capable de prendre la place d'une EPROM ou d'un microcontrôleur dans le montage définitif. Le code exécutable est alors contenu dans une mémoire vive intégrée à l'émulateur. Le système cible vient y lire le programme comme dans une EPROM, tandis que le logiciel de développement peut y pratiquer toutes les modifications souhaitables en un clin d'œil.

Des aller-retours incessants peuvent ainsi être faits entre le code source et le code exécutable, ce qui accélère massivement le processus de développement tout en économisant de multiples effacements d'EPROM.

Il existe des émulateurs spécifiques pour la plupart des microprocesseurs et microcontrôleurs, et également des modèles plus universels capables de prendre en charge différentes familles de composants.

Les réfractaires au langage assembleur peuvent souvent se rabattre sur un langage plus « évolué » tel que le C ou le Pascal. Un logiciel appelé compilateur traduit alors le programme en langage assembleur, facile à transformer en code exécutable par le processus habituel.

Le code ainsi obtenu n'est pas toujours aussi efficace que celui produit par programmation directe en assembleur. Il reste souvent plus encombrant et plus lent, mais des progrès très significatifs ont été accomplis, notamment au départ du langage C.

Figure 4.3.
Un exemple de
programme écrit
en Pascal.

```

var
  timer, frac: word;

procedure interrupt timer0;
begin
  inc(frac);
  if frac=4000 then
  begin
    inc(timer);
    frac:=0
  end;
  TF0:=false
end;

begin
  timer:=0;
  TR0:=true;
  ET0:=true; { enable timer0 interrupts }
  TMO:=2; { Timer0 M1 true: 8 bit autoreload timer }
  TH0:=6; { interrupt every 250 microsecond (if 12MHz clock) }
  frac:=0;
  EA:=true; { enable general interrupt }
  repeat
    (* use timer for something *)
    pl.0:=((timer and 1)=0);
  until false;
end.

```

La figure 4.4 reproduit maintenant, en extrait, le début de la traduction en assembleur réalisée par un compilateur Pascal. Les instructions Pascal d'origine restent présentes sous la forme de « commentaires », tandis que les routines assembleur correspondantes sont rédigées à partir des mnémoniques habituels, exactement comme le ferait un programmeur écrivant directement en langage assembleur.

Ce programme peut donc être soumis tel quel à un logiciel assembleur-linker prévu pour la famille 8051, de façon à obtenir directement ou en deux temps le code exécutable de la figure 4.5 (présenté ici sous la forme d'un fichier INTEL-HEX).

Cette suite d'octets peut être directement « brûlée » dans l'EPROM du microcontrôleur. On dit qu'il s'agit de code

« ROMable », par opposition au code délivré par certains compilateurs moins spécialisés, qui ne peut être exécuté qu'avec l'aide du système d'exploitation d'un micro-ordinateur (par exemple MS-DOS).

```

; MicroControllerPascal compiler, vers. 3.09 copyright (C) KSC 1986, 1990
; With code generation for INTEL 8051
$DEBUG

```

```

NAME      DEMO

USING     3
USING     2
USING     1
USING     0

; timer, frac: word;
?DEMO?D SEGMENT DATA
RSEG      ?DEMO?D
TIMER:    DS      02H
RSEG      ?DEMO?D
FRAC:     DS      02H
;
; procedure interrupt timer0;
; begin

?DEMO?C SEGMENT CODE
RSEG      ?DEMO?C

TOINT:
  PUSH    PSW
  PUSH    ACC
  PUSH    B
  PUSH    DPL
  PUSH    DPH
  MOV     PSW, #010H
; inc(frac);
  INC     FRAC
  MOV     A, #00H
  CJNE    A, FRAC, $+5
  INC     FRAC+01H
; if frac=4000 then
  MOV     A, FRAC+01H
  CJNE    A, #15, $+0BH
  MOV     A, FRAC
  CJNE    A, #160, $+06H
  LJMP    ?L0
  LJMP    ?L1

?L0:
; begin
; inc(timer);
  INC     TIMER
  MOV     A, #00H
  CJNE    A, TIMER, $+5
  INC     TIMER+01H
; frac:=0
; end;
  MOV     FRAC, #LOW(00H)
  MOV     FRAC+01H, #HIGH(00H)

?L1:
; TF0:=false
; end;
  CLR     TF0
  POP     DPH

```

Figure 4.4.
Langage assembleur
généré par le compila-
teur Pascal.

```

:0200000080245A
:03000B0002002CC4
:2000260012009902006EC0D0C0E0C0F0C082C08375D01005237400B523020524E524B40F7A
:2000460008E523B4A00302005202006105217400B521020522752300752400C28DD083D040
:2000660082D0F0D0E0D0D032752100752200D28CD2A9758902758C06752300752400D2AFF7
:200086008E5215401C3940060027401F413929080EF80FE752526D000D001852581C001C0AE
:0200A600002236
:00000001FF

```

Figure 4.5.

Fichier Intel hex du
code exécutable.

L'échantillon de la figure 4.6 montre enfin comment ce code machine pas spécialement explicite peut être remis sous la forme plus lisible de mnémoniques par un logiciel désassembleur. On constate immédiatement que le libellé des instructions ne correspond pas exactement à celui de la figure 4.4, et c'est normal !

Figure 4.6.

Désassemblage du
fichier Intel hex.

```

AMS 8032 PROGRAM lister  DATE 1/2/1980  TIME 1:37  page 1
COPYRIGHT 1989  A. M. S.

PC      MACHN      OPCODE      OPERAND
0026    120099
0029    02006E      LCALL      0099
002C    C0D0        LJMP       006E
002E    C0E0        PUSH       D0
0030    C0F0        PUSH       E0
0032    C082        PUSH       F0
0034    C083        PUSH       82
0036    75D010      PUSH       83
0039    0523        MOV        D0, #10
003B    7400        INC        23
003D    B52302      MOV        A, #00
0040    0524        CJNE       A, 23, 0042
0042    E524        INC        24
0044    B40F08      MOV        A, 24
0047    E523        CJNE       A, #0F, 004F
0049    B4A003      MOV        A, 23
004C    020052      CJNE       A, #A0, 004F
004F    020061      LJMP       0052
0052    0521        LJMP       0061
0054    7400        INC        21
0056    B52102      MOV        A, #00
0059    0522        CJNE       A, 21, 005B
005B    752300      INC        22
005E    752400      MOV        23, #00
0061    C28D        MOV        24, #00
0063    D083        CLR        8D
0065    D082        POP        83
0067    D0F0        POP        82
0069    D0E0        POP        F0
006B    D0D0        POP        E0
006D    32          POP        D0
006E    752100      RETI
0071    752200      MOV        21, #00
0074    D28C        MOV        22, #00
0076    D2A9        SETB       8C
0078    758902      SETB       A9
          MOV        89, #02

```

En fait, le listing de départ utilise des noms de variables et des étiquettes arbitraires, dont le logiciel assembleur calcule les valeurs à la place du programmeur (ou du compilateur). C'est par exemple le compilateur qui a décidé que « PSW » correspondrait au registre « D0 », et « DPL » à l'adresse mémoire « 82 », choix que le désassembleur n'a aucun moyen de connaître...

Volontairement extraits d'un programme pourtant très simple, ces exemples paraîtront certainement bien hermétiques à bon nombre de nos lecteurs, et cela n'a rien de surprenant. On ne peut en effet s'attaquer à la programmation d'un microcontrôleur qu'à la double condition d'avoir appris au préalable un langage de programmation approprié (assembleur, Pascal, ou C), et de posséder une bonne connaissance matérielle du composant.

Programmer correctement un microcontrôleur est nettement plus complexe que personnaliser un GAL ou remplir une EPROM avec le programme d'un séquenceur.

La vocation de cet ouvrage est de présenter la panoplie des composants programmables en général, pas de détailler les mécanismes de programmation de tel ou tel microcontrôleur. Il existe d'excellents livres traitant uniquement de ces questions, auxquels nous renvoyons nos lecteurs désireux de les approfondir.

LES PROGRAMMATEURS POUR MICROCONTRÔLEURS

L'issue normale de tout le processus de développement que nous venons de présenter est bien évidemment le transfert du programme dans l'EPROM du microcontrôleur.

La marche à suivre diffère notablement d'un composant à l'autre : passage par une EPROM intermédiaire et utilisation d'un accessoire « chargeur » pour le 68705P3, utilisation d'un programmeur spécifique pour le PIC 16C54, simple adjonction d'un adaptateur sur un programmeur d'EPROM pour les 8751 et 8753, ou même téléchargement « en circuit » pour le PIC16C84.

Les programmeurs « universels » du commerce peuvent généralement supporter un assez large choix de microcontrôleurs, mais pas forcément exhaustif malgré leur prix confortable. Cela donne à réfléchir...

LES MICROCONTRÔLEURS PIC

Le terme microcontrôleur a longtemps évoqué un boîtier à 40 ou au moins 28 broches, offrant de multiples ports d'entrée-sortie, relativement cher, pas toujours très rapide, et assez délicat à programmer sans un coûteux système de développement. Ce n'est évidemment pas l'idéal pour un certain nombre d'applications de grande diffusion devant allier miniaturisation, performances, faible coût, et rapidité de mise sur le marché.

Parmi les microcontrôleurs « de bas de gamme » aujourd'hui très populaires, la famille PIC 16CXX de Microchip possède une belle panoplie d'atouts... Dans notre contexte, le terme « bas de gamme » ne doit pas être perçu avec un sens péjoratif : il s'applique en effet à des produits simples et bon marché faisant cependant appel à des techniques de pointe. En matière de microcontrôleurs, on entendra par « bas de gamme » un composant équipé d'un nombre réduit de lignes d'entrée-sortie, disponible en boîtier plastique OTP à faible nombre de broches, pouvant se passer de quartz au profit d'un résonateur céramique ou d'un réseau RC, et facile à programmer. On rencontrera de tels produits dans les téléphones portables, les récepteurs de radio-messagerie, les systèmes de contrôle d'accès, les télécommandes codées, les capteurs « intelligents », dans toute une gamme d'équipements de grande consommation à commencer par l'automobile et l'électroménager, et jusque dans les kits où ils pourront facilement se faire passer pour des circuits intégrés ordinaires mais non identifiables si on prend la précaution d'en effacer le marquage.

En fait, leur domaine de prédilection sera toutes les applications dans lesquelles il est trop complexe, trop coûteux, ou trop encombrant de mettre en œuvre de la logique câblée, mais dans lesquelles il serait aussi trop complexe, trop coûteux, ou trop encombrant d'incorporer un microcontrôleur conventionnel. Des applications nouvelles, donc, et par conséquent des produits souvent très innovateurs, mais aussi un surcroît de performances et de possibilités pour toute une gamme de produits existants que le passage au microcontrôleur permet de perfectionner considérablement.

Les microcontrôleurs PIC 16CXX se situent précisément à ce « carrefour » stratégique entre logique traditionnelle, réseaux programmables, et microprocesseurs. Ils s'imposeront d'eux-mêmes toutes les fois que leur utilisation permettra de mettre sur le marché un produit plus performant et plus petit, plus rapidement et pour un coût inférieur.

La gamme PIC 16CXX est construite autour de deux configurations matérielles de base, à 12 ou 20 lignes d'entrée-sortie respectivement.

Le produit le plus simple, logé dans un boîtier DIP ou SOIC à 18 broches, est le PIC 16C54. Il possède 12 lignes d'E/S, une RAM de 32 octets, et une EPROM de 512 × 12 bits (ce qui est déjà en soi une originalité).

Compatible broche pour broche, le PIC 16C56 est équipé d'une EPROM de capacité double.

Disponible en boîtier DIP ou SOIC à 28 broches, le PIC 16C55 dispose de 20 lignes d'E/S, de 32 octets de RAM, et de 512 × 12 bits d'EPROM.

Sa variante PIC 16C57 offre une EPROM de capacité quatre fois supérieure.

Le PIC16C71, pour sa part, est un composant plus récent, sensiblement amélioré notamment par l'ajout d'entrées analogiques (CAN incorporé), et utilisant des instructions de 14 bits au lieu de 12, programmables dans une ROM de 1 024 adresses.

Encore plus récent mais déjà très répandu, le PIC16C84 est réalisé en technologie EEPROM. C'est dire que malgré sa présentation en boîtier plastique, il peut être effacé (électriquement) en un clin d'œil et aussitôt reprogrammé. Une partie de sa mémoire non volatile est même disponible pour des données qui ne doivent pas être perdues lorsque l'alimentation disparaît.

La présentation la plus courante des versions EPROM est le boîtier plastique (OTP), mais il existe des versions en boîtier céramique à fenêtre (UVPROM) utilisées en développement. Un service de programmation en usine est disponible (QTP), qui remplace avantageusement, compte tenu de son coût, le masquage de ROM.

Autant dire que le passage du prototype à la série pourra se faire rapidement et sans frais notables, et que l'impact d'éventuelles modifications ultérieures restera extrêmement limité.

Tous ces composants réalisés en technologie CMOS s'alimentent en 5 V ou en 3 V (versions à faible consommation), consomment entre 15 µA et 2 mA selon la fréquence d'horloge, possèdent un mode *standby* ramenant leur consommation à moins de 3 µA, et sont disponibles dans différentes configurations au niveau de l'oscillateur d'horloge (0 à 20 MHz).

Ils sont donc parfaitement aptes à fonctionner sur piles, dans des applications portatives ou même de poche.

Tous les PIC disposent en outre d'un compteur-timer intégré, dont l'entrée d'horloge est disponible sur une broche réservée, mais qui peut aussi fonctionner sur l'horloge principale. Enfin, un chien de garde programmable est disponible.

L'originalité majeure des PIC est leur architecture RISC (*Reduced Instruction Set Computer*) qui, par rapport à la configuration classique dite « CISC », présente notamment les avantages suivants :

- code plus compact (d'un facteur 2),
- rapidité accrue (d'un facteur 4),
- accélération du développement (de près de 30 %).

Au lieu des 50 à 110 instructions à plusieurs octets et plusieurs cycles d'horloge des microcontrôleurs conventionnels, le langage des PIC16C5X se limite à 33 instructions exécutables en un seul cycle (sauf instructions de branchement à deux cycles) et tenant sur un seul mot mémoire (de 12 bits au lieu de 8). Pour sa part, le PIC16C71 est doté de 35 instructions.

L'amélioration est d'autant plus appréciable que beaucoup de programmeurs ne se servent guère des instructions les plus complexes des CISC, notamment parce que celles-ci sont par trop différentes d'un processeur à l'autre.

Certes, les logiciels écrits en langage de haut niveau (C), puis compilés, en font usage mais souvent au prix d'un code plus encombrant et plus lent que nécessaire.

Avec un jeu d'instructions réduit à l'essentiel, il devient paradoxalement plus facile et plus rapide d'écrire directement en assembleur des programmes compacts et efficaces. Ce principe est supporté par une architecture particulière, dite Harvard, dans laquelle deux bus distincts sont affectés aux données et aux instructions. Cela élimine radicalement le goulot d'étranglement classique dans les architectures CISC.

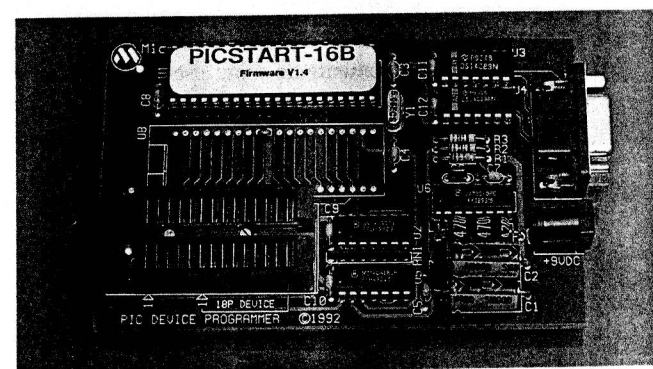
Parallèlement, un système dit de Pipe-Line permet l'exécution d'une instruction pendant que l'on va chercher la suivante, tirant ainsi le maximum possible de chaque cycle d'horloge. Il résulte de tout cela des performances pouvant atteindre les 5 MIPS (Millions d'Instructions Par Seconde), ce qui rend possibles des approches très originales de certains problèmes avantageux à traiter en temps réel.

Avec un PIC, on peut par exemple souvent remplacer d'encombrantes tables de données par un algorithme de calcul rapide, presque comme on le ferait, toutes proportions gardées, avec un DSP (*Digital Signal Processor*).

Les microcontrôleurs PIC sont suffisamment originaux pour justifier l'existence d'outils de développement spécifiques, qui peuvent fort heureusement être réunis à très peu de frais.

Le succès commercial de microcontrôleurs bon marché et « sympathiques » à programmer étant en effet incompatible avec des systèmes de développement par trop coûteux, les fabricants ont compris qu'il fallait faire un effort de « démocratisation ».

Motorola a choisi de mettre gratuitement les logiciels nécessaires à la disposition de ses clients, laissant à ceux-ci le soin de construire eux-mêmes un programmeur très simple. SGS-Thomson, de son côté, a préféré commercialiser un « kit de développement » complet, mais à un prix attractif.



Microchip applique une politique qui combine adroitement ces deux approches : un starter kit est disponible à un prix somme toute raisonnable, compte tenu de ce qu'il contient. Baptisé PICSTART-16B, il réunit en effet un programmeur dédié (pour PC), les logiciels assembleur, simulateur et programmeur, une documentation complète, et quelques PIC effaçables aux ultra-violets.

Parallèlement, des logiciels pratiquement équivalents sont disponibles gratuitement par téléchargement (BBS ou Internet) tandis qu'après quelques réticences, les algorithmes de programmation ont finalement été rendus publics (notamment sur un très intéressant CD-ROM). Nous verrons ainsi, au chapitre 5, à quel point il peut être facile de réaliser, par exemple, un programmeur de PIC16C84.

Le logiciel de pilotage du programmeur Picstart est agréable et simple à utiliser, grâce à une présentation à « fenêtres » rappelant Windows sans pour autant en hériter la lenteur et l'encombrement mémoire. Il permet de charger un fichier hexadécimal issu de l'assembleur, éventuellement de le modifier ponctuellement, puis de le « brûler » dans un PIC après avoir configuré les fusibles optionnels : chien de garde, protection contre la copie, type d'oscillateur, identification. Après programmation, une relecture de contrôle est exécutée, mais on peut aussi lire ou vérifier un PIC à tout moment et naturellement contrôler la virginité d'un composant neuf ou effacé.

Les logiciels assembleur et simulateur ne paient pas de mine au premier abord. Fonctionnant en mode « ligne de commande », ils évoquent les temps héroïques de l'informatique, lorsque tout le travail se faisait sur des téléimprimeurs ! À y regarder de plus près, on se rend cependant vite compte qu'il s'agit de puissants outils n'ayant pas grand chose à envier (à part les fioritures sur l'écran) à des produits dont le prix peut atteindre cinq à dix fois celui du kit complet ! L'utilisateur devra donc faire l'effort d'apprendre un certain vocabulaire d'ordres abrégés (à moins qu'il ne se contente de compulser l'aide en ligne aussi souvent que nécessaire) plutôt que de se contenter de cliquer avec sa souris. Mais à notre sens, c'est cela (entre autres !) qui fait la différence entre un véritable programmeur et un simple employé de bureau...

Le simulateur permet de faire tourner artificiellement un programme préalablement traité par l'assembleur, plusieurs dizaines à plusieurs centaines de fois moins vite qu'en réalité. Cela, associé à la possibilité de poser des points d'arrêt et de visualiser le contenu des registres, permet de figurer la mise au point de programmes très « pointus » à un cycle d'horloge près.

Un exemple très représentatif figure notamment dans notre ouvrage *PC et cartes à puces*, paru dans cette même collection.

1	Introduction aux composants programmables	9
2	Les mémoires programmables	13
3	Les réseaux logiques programmables	57
4	Les microcontrôleurs	95

5 RÉALISATION DE PROGRAMMATEURS

Un programmeur d'EPROM	112
Un lecteur d'EPROM	122
La programmation et la lecture des RAM ZEROPOWER	131
Une alimentation de sécurité pour programmation d'EPROM	134
La programmation des ispGAL 22V10	138
La programmation des ispLSI 1016 et 2032	145
Un programmeur de PIC	146
La programmation des EEPROM série	151
Le logiciel de programmation	154

6	La disquette du livre	155
---	-----------------------	-----

Voici venu le moment de passer à la pratique en programmant pour de bon des composants ! Pour ce faire, il faut évidemment un matériel approprié, autrement dit un ou plusieurs programmeurs. Il ne fait aucun doute que la démarche la plus efficace et la plus souple consiste à employer un programmeur adaptable à un PC, et quelques uns des puissants logiciels développés à cet effet.

UN PROGRAMMATEUR D'EPROM

Le micro-ordinateur compatible PC est désormais un équipement très répandu et peu coûteux, aux innombrables applications.

Contrairement à ce que l'on pourrait penser, des montages extrêmement simples et économiques suffisent pour lui permettre de programmer et lire des EPROM et bien d'autres composants programmables, sans intervenir en quoi que ce soit à l'intérieur de la machine. Tout cela grâce à des programmes écrits dans des langages courants et souvent longs de quelques dizaines de lignes seulement !

Nous savons en effet qu'à la base, programmer une EPROM (ou un composant à base d'EPROM) est une opération simple : il suffit de sélectionner tour à tour toutes les adresses auxquelles quelque chose doit être écrit et, en présence de la donnée à programmer (en général un mot de huit bits), d'appliquer une impulsion bien calibrée à une broche spéciale. Le tout bien sûr en respectant les tensions d'alimentation spécifiées par le fabricant.

Les données à programmer peuvent être disponibles déjà dans une EPROM (pas forcément du même type d'ailleurs), sur disquette ou sur papier, voire dans la tête de l'opérateur !

Ce qui complique les choses est que d'un type d'EPROM à l'autre, brochage, tensions d'alimentation, et cycle de programmation affichent des différences, et pas toujours mineures.

S'il faut que le programmeur soit très « intelligent » et donc qu'il tienne compte automatiquement de tous les cas possibles, ce sera nécessairement un appareil complexe, à « support configurable par logiciel ».

S'il faut qu'il soit extrêmement rapide, il n'y a guère d'autre solution que de l'interfacer directement sur les bus d'adresse et de données de l'ordinateur hôte, à l'aide de circuits du genre « PIA ».

Pour notre part, nous considérons qu'un électronicien digne de ce nom est très capable de brancher quelques cordons sans se tromper, et qu'il n'est pas rédhibitoire de devoir attendre quelques instants que la programmation soit achevée. Notre idée consiste donc à utiliser la prise d'imprimante parallèle, disponible sur tous les PC, pour connecter notre programmeur.

Ses huit lignes de données permettent d'amener directement les octets à programmer, tandis que la ligne de STROBE peut facilement véhiculer les impulsions de programmation, dont la durée sera réglée par logiciel (pas besoin de monostable de précision !). Quelques lignes d'entrée (au moins un ACK et un BUSY) sont par ailleurs à notre disposition. Dans un deuxième temps, nous nous en servons pour lire les EPROM, à des fins de duplication ou de vérification voire, pourquoi pas, de modification.

Pas moyen toutefois de faire passer directement un bus d'adresses par ce chemin trop « étroit » pour lui. Qu'à cela ne tienne ! Dans l'immense majorité des cas, on programme une EPROM à partir de sa première adresse jusqu'à sa dernière, ou plus rarement de façon partielle, mais « séquentiellement », dans l'ordre croissant des adresses. Un simple compteur utilisant les impulsions de programmation comme signal d'horloge pourra très facilement « balayer » tous les états possibles du bus d'adresses, à condition qu'on le mette à zéro en synchronisme avec le logiciel. Ce n'est pas bien difficile...

Pour ce qui est des alimentations, tout laboratoire normalement équipé contient ce qu'il faut pour produire du 5 V et une seconde tension pouvant aller de 12 à 25 V, le tout sous quelques dizaines de milliampères. On pourra au besoin construire une petite alimentation spécialisée, mais cela n'a rien d'obligatoire.

La figure 5.1 détaille la solution pratique retenue : un compteur CMOS 4040 et un sextuple inverseur 4049 sont les deux seuls composants actifs nécessaires !

Huit résistances de 470 Ω (ou un réseau DIL) sont intercalées sur le bus de données pour éviter tout problème en cas de conflit : elles limitent le courant pouvant circuler si la mémoire se trouvait mise accidentellement en mode lecture.

Les huit lignes de données ainsi protégées rejoignent directement le support, car leur emplacement est le même pour toutes les mémoires conformes au standard Byte-wide.

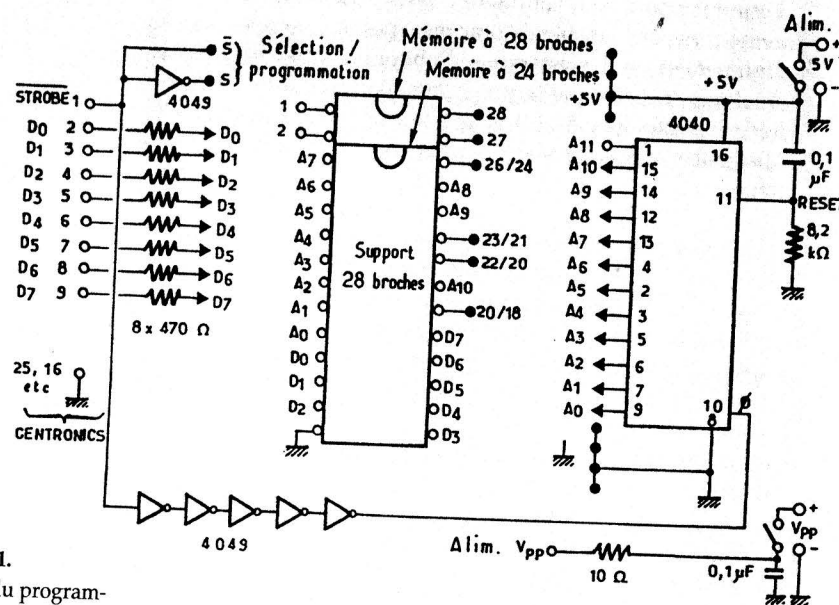


Figure 5.1.
Schéma du programmeur.

Le signal de STROBE et son complément sont mis à la disposition de l'utilisateur sur deux bornes, afin qu'une impulsion de la bonne polarité puisse être dirigée sur la mémoire, selon son type.

Complémenté et légèrement retardé par cinq inverseurs en cascade, ce même STROBE attaque l'entrée d'horloge du 4040, dont la remise à zéro est assurée par un réseau RC lors de la mise sous tension.

Douze lignes d'adresse sont disponibles en sortie du 4040, ce qui suffit pour des mémoires allant jusqu'à 4 kilo-octets ($2^{12} = 4096$), c'est-à-dire jusqu'à la 2732. Les lignes A0 à A10, communes à toutes les EPROMS de la 2716 à la 27256 et même au-delà, rejoignent directement le support. A11 est offerte sur une borne permettant son raccordement « en volant » à la broche 23/21 du support, à partir de la 2732.

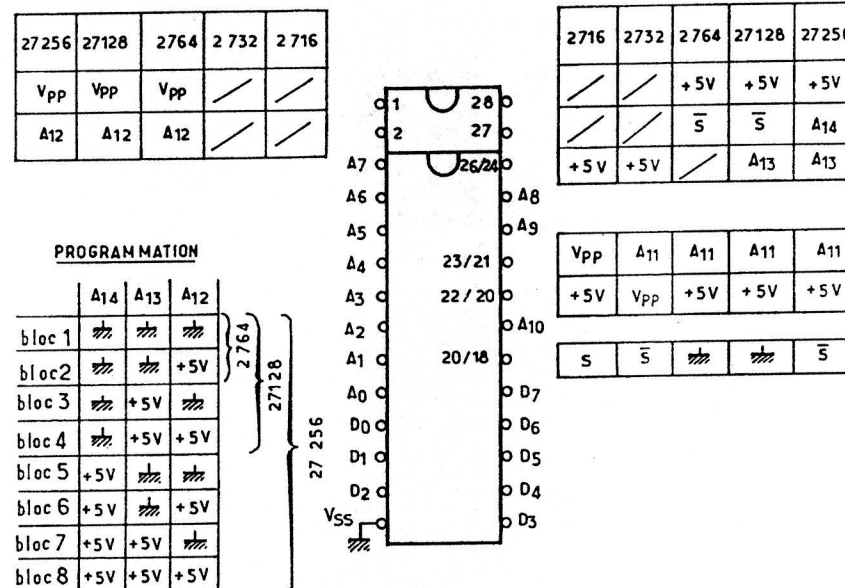
Pour exploiter les lignes A12 à A14, il faut utiliser les broches 1, 2, 27, et 28 du support, qui restaient vides avec les 2716 et 2732.

La figure 5.2, qui définit les connexions à effectuer pour programmer les EPROM les plus courantes, montre comment des manœuvres relativement simples de cordons permettent de programmer facilement des 2764 en deux blocs de 4 kilo-octets,

et de façon plus fastidieuse, les 27128 en quatre fois et les 27256 en huit fois. Cette procédure serait évidemment bien trop lourde pour un usage régulier de mémoires de capacité utile supérieure à 8 kilo-octets, mais il est appréciable de disposer de cette facilité « au cas où », moyennant l'achat d'un support à 28 broches au lieu de 24 ! Rappelons en effet qu'il n'est nullement exceptionnel de rencontrer des 27C512 ou même des 27C010 ne contenant guère que l'équivalent d'une 2716...

Enfin, deux interrupteurs permettent d'appliquer et de couper les deux tensions d'alimentation V_{cc} et V_{pp} selon les invitations du logiciel.

Figure 5.2.
Adaptation aux principales EPROM.



On sait en effet que la présence de V_{pp} en l'absence de V_{cc} est fatale à la mémoire, en quelques fractions de seconde ! Nous persistons cependant à penser que nos lecteurs sont capables de respecter une telle procédure ! Un montage aussi simple tient sans difficulté sur un circuit imprimé conforme au tracé de la figure 5.3.

L'originalité réside dans le fait que le côté composants (figure 5.4) est assez peu mis à contribution : on y trouve les deux circuits intégrés, les résistances et condensateurs, un strap, et

enfin les départs de câbles (alimentations V_{cc} et V_{pp} , et câble plat à dix conducteurs en provenance de la prise normalisée Centronics identique à celle d'une imprimante).

Figure 5.3.
Circuit imprimé du
programmeur.

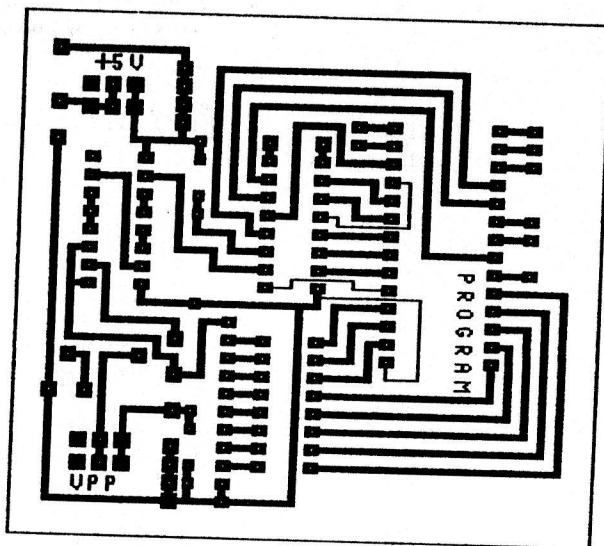
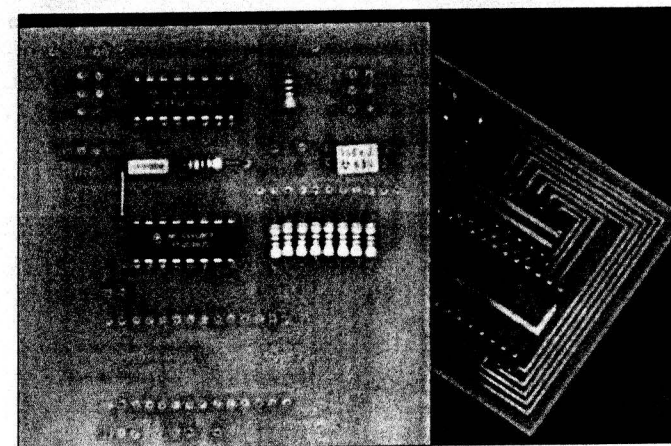
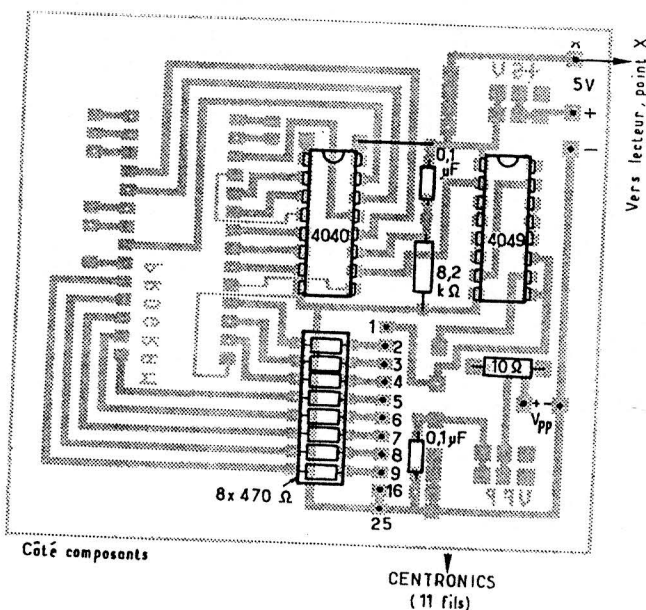


Figure 5.4.
Plan de câblage du
côté composants.

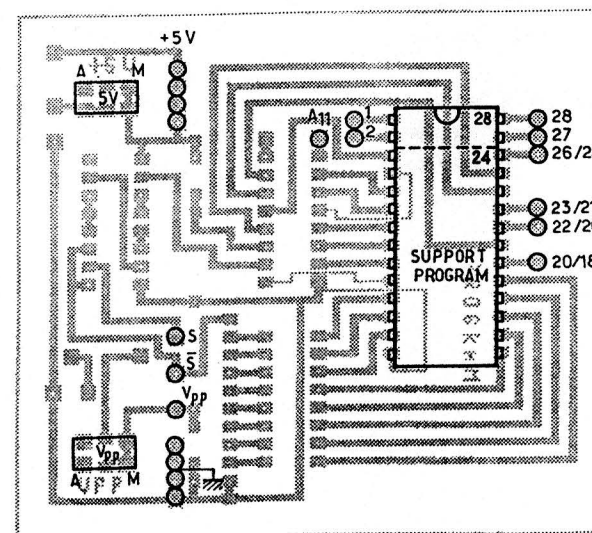
Liste des composants
du programmeur
pour PC

- 1 x CD 4040.
- 1 x CD 4049.
- 1 x 10 Ω .
- 1 x 8,2 k Ω .
- 8 x 470 Ω ou 1 réseau DIL.
- 1 x 0,1 μ F.
- 1 support «tulipe» 28 broches.
- 2 interrupteurs unipolaires.
- 1 barrette sécable «tulipe».
- 1 connecteur Centronics (modèle imprimante).
- Câble plat 11 conducteurs.



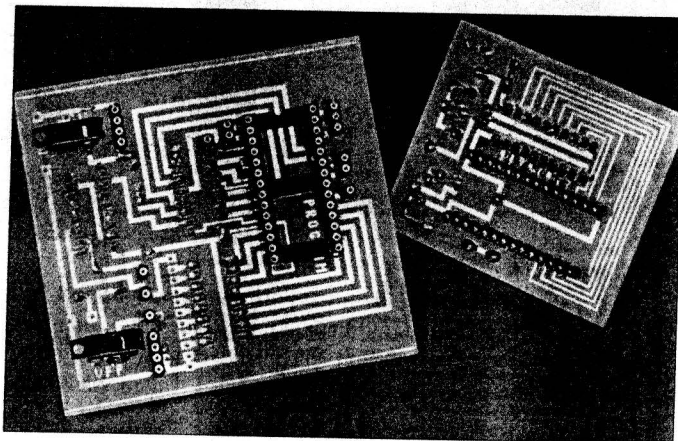
C'est côté cuivre (figure 5.5) que l'on soudera le support à 28 broches (de type tulipe ou mieux à force d'insertion nulle), les deux interrupteurs, et surtout 20 contacts tulipe extraits d'une « barrette sécable » ou, à défaut, récupérés sur un vieux support de circuit intégré.

Figure 5.5.
Plan de câblage du
côté cuivre.



Ce sont ces tulipes qui vont faire office de douilles pour la configuration matérielle du support : au lieu de cordons à fiches banane, on utilisera des morceaux de fil de câblage rigide de 6/10 (diamètre impératif) dénudés à chaque bout sur 3 à 4 mm. C'est beaucoup moins encombrant, au moins aussi pratique, et d'un prix imbattable !

Un avantage supplémentaire de cette disposition est que la carte peut être fixée tout contre le panneau supérieur d'un coffret pupitre, avec de simples écrous en guise d'entretoises. Support, tulipes et interrupteurs affleurent juste, ce qui est à la fois esthétique, pratique, et sûr.



Les logiciels

La conception simplifiée à l'extrême de cette partie matérielle entraîne évidemment le report sur le logiciel des principaux problèmes à résoudre.

Les programmes doivent en particulier gérer directement les ports d'entrée-sortie reliés à la prise d'imprimante, et donc communiquer avec les adresses qui leur sont affectées. Tous les logiciels publiés ici sont destinés à fonctionner en liaison avec le port « LPT1: » d'un « vrai » compatible PC, et devraient donc subir des modifications pour « tourner » le cas échéant sur d'autres configurations.

Deux versions sont proposées pour chaque programme : l'une écrite en BASIC, langage généralement livré d'origine avec tous les PC, et l'autre développée en Turbo-Pascal. Ce langage très populaire est considérablement plus rapide que le BASIC, mais n'est pas livré en « standard » avec le premier PC venu. La disquette d'accompagnement de cet ouvrage réunit les sources BASIC et Turbo-Pascal, ainsi que les exécutables (fichiers .EXE) correspondants. On n'aura donc besoin du compilateur Borland que pour procéder à d'éventuelles modifications.

Le logiciel PROGRAM (PROGRAM.BAS à la figure 5.6 et PROGRAM.PAS à la figure 5.7) va chercher les données à programmer dans un fichier « texte » présent sur disquette (avec une extension « .ROM » implicite), dans lequel chaque octet est représenté sous la forme d'une valeur décimale en clair (forme déjà présentée à la figure 2.4, et connue sous le nom de « free format decimal »).

```

10 REM ---- PROGRAM.BAS ----
20 CLS:PRINT"COUPER LES ALIMENTATIONS"
30 PRINT"nom du fichier à brûler en EPROM ?"
40 INPUT F$:F$=F$+".ROM"
50 OPEN"1",#1,F$
60 DIM M(4096)
70 CLS:PRINT:PRINT"---- LECTURE FICHIER EN COURS ----"
80 F=0
90 IF EOF(1) THEN 130
100 INPUT#1,M(F)
110 F=F+1
120 GOTO 90
130 CLOSE#1:CLS
140 PRINT"DUREE IMPULSION en millisecondes ?"
150 INPUT M
160 IF M>50 THEN M=50
170 IF M<5 THEN M=5
180 CLS:PRINT"connecter une EPROM vierge"
190 PRINT"d'au moins ";F;" octets"
200 PRINT"puis presser ENTER"
210 INPUT Z$:CLS
220 PRINT"appliquer le +5V, puis presser ENTER"
230 INPUT Z$
240 PRINT"appliquer le Vpp, puis presser ENTER"
250 PRINT:PRINT"ATTENTION A LA VALEUR DE Vpp!"
260 INPUT Z$
270 CLS:PRINT:PRINT"----- PROGRAMMATION EN COURS -----"
280 FOR G=0 TO F-1
290 D=M(G)
300 OUT 888,D
310 OUT 890,1
320 FOR T=1 TO M:NEXT T
330 OUT 888,255
340 OUT 890,0
350 NEXT G
360 PRINT:PRINT:PRINT:BEEP:CLS
370 PRINT"COUPER LE Vpp puis presser ENTER"
380 INPUT Z$:PRINT
390 OUT 888,0
400 PRINT"COUPER LE +5V puis presser ENTER"
410 INPUT Z$
420 CLS:PRINT"RETIRER L'EPROM"
430 PRINT:PRINT"RECOMMENCER ? O/N + ENTER"
440 INPUT Z$
450 IF Z$="O" OR Z$="o" THEN 180
460 END
470 REM (c)1991 Patrick GUEULLE

```

Figure 5.6.
Version BASIC
du logiciel
PROGRAM.

Figure 5.7.
Version Turbo-
Pascal du logiciel
PROGRAM.

```

program PROGRAM;
uses crt;
var m:array[0..4095] of integer;
    h:string[12];
    f:integer;
    e:boolean;
    g:integer;
    a:text;
    t:integer;
    n:string[11];
procedure Init;
begin
  clrscr;
  port[888]:=0;
  writeln('ATTENTION! COUPER LES ALIMENTATIONS');
  writeln;
end;
procedure Lecture;
begin
  writeln('NOM DU FICHIER A ECRIRE EN EPROM ?');
  readln(h);
  h:=h+'.rom';
  assign(a,h);
  reset(a);
  clrscr;
  writeln('----- LECTURE FICHIER EN COURS -----');
  f:=0;
  repeat
    read(a,m[f]);
    e:=seekeof(a);
    f:=f+1;
  until e=true;
  close(a);
end;
procedure Eeprom;
begin
  clrscr;
  writeln('DUREE IMPULSION en millisecondes ?');
  readln(t);
  if t>50 then t:=50;
  if t<5 then t:=5;
  clrscr;
  writeln('CONNECTER UNE EPROM VIERGE');
  writeln('de ',f,' octets au moins');
  writeln('puis presser ENTER');
  readln;
  clrscr;
  writeln('APPLIQUER LE +5V puis presser ENTER');
  readln;
  writeln('APPLIQUER LE Vpp puis presser ENTER');
  writeln('ATTENTION A LA VALEUR DE Vpp!');
  readln;
  clrscr;
  writeln('----- PROGRAMMATION EN COURS -----');
  for g:=0 to f-1 do
  begin
    port[888]:=m[g];
    port[890]:=1;
    delay(t);
    port[888]:=255;
  end;
end;

```

```

port[890]:=0;
end;
clrscr;
sound(440);
delay(1000);
nosound;
writeln('COUPER LE Vpp puis presser ENTER');
readln;
port[888]:=0;
writeln('COUPER LE +5V puis presser ENTER');
readln;
clrscr;
writeln('RETIRER LA MEMOIRE');
writeln;
writeln('RECOMMENCER ? O/N + ENTER');
readln(n);
if (n='o') or (n='O') then eeprom;
end;
begin
  init;
  lecture;
  eeprom;
  writeln('TERMINE');
end.
(* COPYRIGHT 1990 Patrick GUEULLE *)

```

En les lisant, il construit un tableau de 4 096 «cases» dans lequel il pourra aller les chercher lors de la programmation proprement dite, sans avoir à subir à chaque accès disque une interruption capable d'endommager l'EPROM par allongement intempestif de l'impulsion calibrée.

Ce fichier pourra être composé directement avec n'importe quel éditeur de texte, ou obtenu par conversion d'un fichier utilisant un autre «format» : binaire, hexadécimal, etc.

Le logiciel de démonstration du programmeur MQP, reproduit sur la disquette d'accompagnement de cet ouvrage, contient un excellent éditeur-transcodeur de fichiers (PDED) parfaitement adapté à ce genre de travail. Offert en shareware, il suppose l'accomplissement d'une formalité d'enregistrement en cas d'utilisation régulière (voir fichier READ.ME pour les détails).

À défaut, le programme «TRANS.BAS» de la figure 5.8 permet la conversion entre fichiers décimaux et binaires ou vice versa.

Si le programme Turbo-Pascal produit de lui-même une impulsion de programmation parfaitement calibrée, la version BASIC peut nécessiter un « réglage » en fonction de la fréquence d'horloge du PC employé. On contrôlera donc à l'oscilloscope la durée réellement obtenue en demandant 50 ms, et en cas de décalage (probable sur tout PC supérieur au 286), on ajoutera une ligne 175 appliquant un coefficient de correction.


```

10 REM ----- TRANS.BAS -----
20 CLS:PRINT"NOM FICHIER SOURCE ? (avec extension)"
30 INPUT N$
40 PRINT:PRINT"NOM FICHIER DE DESTINATION ? (avec extension)"
50 INPUT F$
60 PRINT:PRINT
70 PRINT"Transformation fichier décimal en fichier binaire: ---> 1"
80 PRINT"Transformation fichier binaire en fichier décimal: ---> 2"
90 PRINT:PRINT"      + ENTER"
100 INPUT Z$
110 IF Z$="2" THEN 230
120 IF Z$<>"1" THEN 100
130 OPEN N$ FOR INPUT AS #1
140 OPEN "r",#2,F$,1
150 FIELD#2,1 AS A$
160 CLS:PRINT"Transformation en cours: ";N$;" en ";F$
170 IF EOF(1) THEN 210
180 INPUT#1,M:N$=CHR$(M)
190 RSET A$=M$:PUT#2,F+1
200 F=F+1:GOTO 170
210 CLS:PRINT N$;" est transformé en ";F$
220 END
230 OPEN F$ FOR OUTPUT AS #2
240 OPEN "r",#1,N$,1
250 FIELD#1,1 AS A$
260 CLS:PRINT"Transformation en cours: ";N$;" en ";F$
270 FOR F=1 TO LOF(1)
280 GET#1,F
290 PRINT#2,ASC(A$);
300 NEXT F
310 GOTO 210
320 REM (c)1991 Patrick GUEULLE

```

Figure 5.8. Utilitaire de transcodage de fichiers.

UN LECTEUR D'EPROM

Même si le programmeur précédemment décrit peut fort bien fonctionner seul, son jumelage avec un lecteur s'impose. D'abord pour faire de la copie d'EPROM existantes ou des transferts entre types différents (quatre 2716 dans une seule 27C64, par exemple), mais aussi à des fins de vérification : test de virginité des EPROM neuves ou effacées (les surprises ne sont pas rares !), contrôle après programmation ou lors d'un dépannage, etc.

Beaucoup de programmeurs sont en même temps lecteurs, les deux opérations se faisant habituellement sans changer la mémoire de support. Avec la conception retenue pour notre programmeur (configuration du support par cordons enfichables), ce procédé serait à peu près inapplicable et en tout cas bien trop fastidieux. Il est donc préférable à tous points de vue de construire séparément le programmeur et le lecteur, même si on les réunit dans un seul boîtier et sur un même connecteur de l'ordinateur.

La communication entre les deux sections de l'appareil se fera par l'intermédiaire du fichier décimal « .ROM » enregistré sur disquette, cet archivage forcé constituant par ailleurs une très réelle sécurité pour les données tout en facilitant considérablement leur inspection, leur duplication, ou leur modification. Un tel fichier peut en effet être « travaillé » à volonté à l'aide de n'importe quel éditeur de texte.

Comme pour le programmeur, nous avons décidé d'utiliser le port d'imprimante parallèle « LPT1: » pour relier le lecteur au PC « hôte », bien qu'il s'agisse d'une sortie.

On ne dispose ainsi que d'un nombre de lignes d'entrée inférieur à huit : ACK, BUSY, PE (*paper end*), et pas grand chose de plus dans le cas d'un port parallèle « ordinaire ».

Nous n'avons donc guère le choix : il va falloir « sérialiser » les données lues dans l'EPROM, c'est-à-dire entrer leurs huit bits à la queue leu leu sur le fil ACK ou BUSY.

La figure 5.9 montre comment cette technique peut être mise en œuvre avec très peu de composants : un compteur 4040 pour faire défiler les adresses comme dans le programmeur, et un « sélecteur de données » 4512.

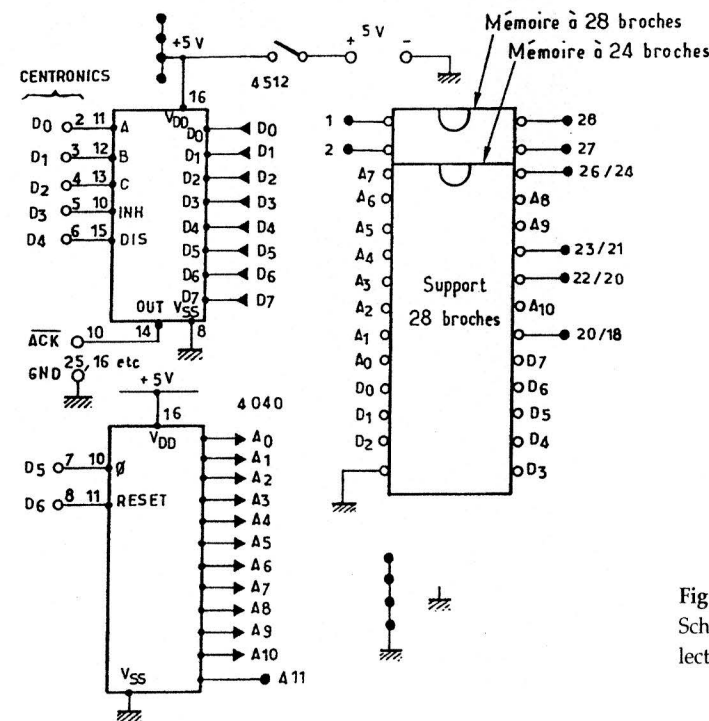


Figure 5.9. Schéma du lecteur.

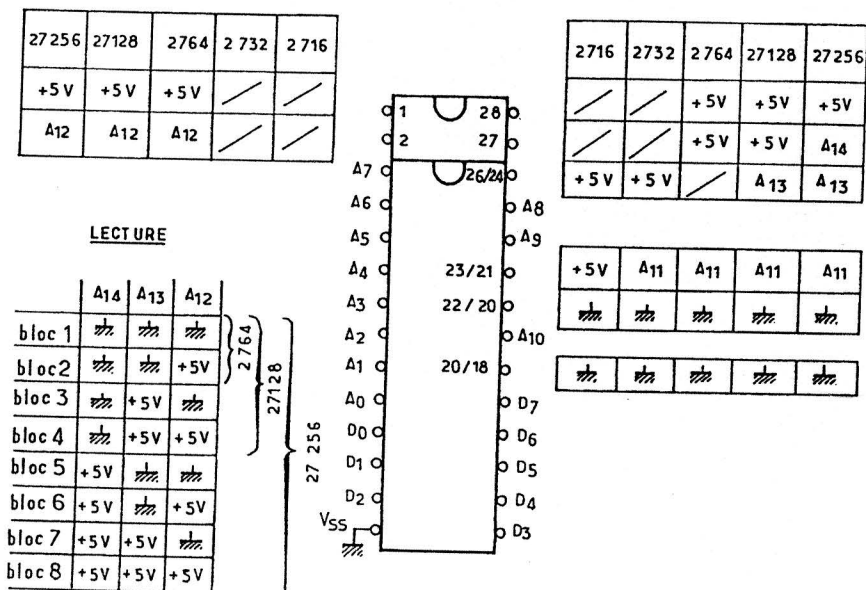
Trois des huit lignes de sortie de la prise parallèle servent à indiquer au 4512 lequel des huit bits du bus de données de la mémoire il doit «aiguiller» vers la ligne d'entrée ACK (broche 10). Deux autres de ces lignes de sortie servent respectivement à mettre le compteur à zéro, et à le faire avancer lorsqu'il faut changer d'adresse, tandis qu'encore deux autres lignes desservent les entrées d'inhibition du 4512. Nous n'en avons pas l'usage pour le moment et les maintiendrons donc logiciellement à zéro, mais cela préserve les possibilités d'évolution du montage.

Évidemment, en fonctionnement, il faut que le logiciel prenne le temps de lire tour à tour ces huit bits, puis de les réassembler en un octet, cela pour chaque adresse. Ce n'est pas très rapide en BASIC, mais nettement plus en Turbo-Pascal.

Une fois encore, nous avons amené au support d'EPROM les lignes communes à tous les types courants, et prévu des contacts «tulipe» pour celles qu'il faut personnaliser. Attention, les connexions ne sont évidemment pas les mêmes en lecture qu'en programmation, et l'alimentation V_{pp} doit être ramenée à 5 volts.

La figure 5.10 rassemble les données nécessaires à la mise en œuvre des types les plus courants d'EPROM, mais il n'est pas interdit de consulter les data books pour lire des mémoires mortes d'autres types : ROM masquées, PROM à fusibles, EEPROM, FLASH, etc.

Figure 5.10.
Adaptation aux
principales EPROM.



Tout le montage tient sur le petit circuit imprimé de la figure 5.11, qui sera câblé selon la même technique que celui du programmeur : circuits intégrés, strap, et câble plat à dix conducteurs côté composants selon la figure 5.12, support et contacts «tulipe» extraits d'une barrette sécable côté cuivre (figure 5.13).

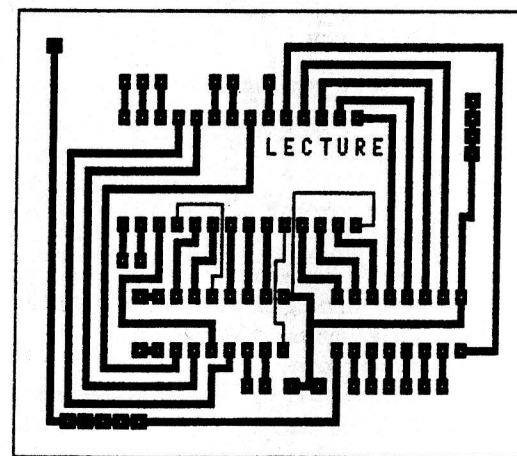


Figure 5.11.
Tracé du circuit
imprimé du lecteur.

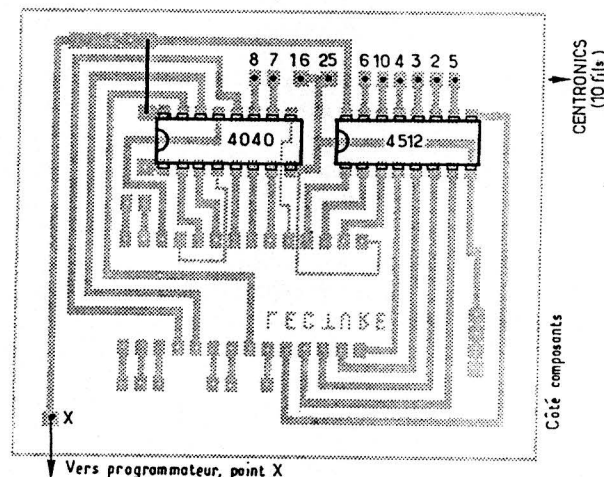
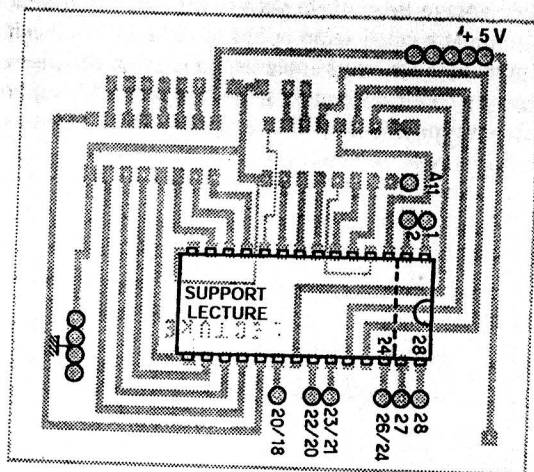


Figure 5.12.
Câblage du côté
composants.

Liste des composants
du lecteur pour PC

- 1 × CD 4040.
- 1 × CD 4512.
- 1 support «tulipe» 28 broches.
- 1 barrette sécable «tulipe».
- Câble plat 10 conducteurs.

Figure 5.13.
Câblage du
côté cuivre.



Le câble plat pourra être soudé sur le même connecteur Centronics que celui du programmeur, c'est-à-dire en parallèle pour ce qui est des lignes de données D0 à D6, de STROBE, et de masse. Il n'y aura donc pas de commutation à effectuer entre les modes « lecture » et « programmation » : il suffira de changer de logiciel et de transporter l'EPROM d'un support à l'autre, hors tension évidemment, mais l'ordinateur pouvant rester en service. Bien que des protections soient prévues, il est vivement déconseillé que les supports de lecture et de programmation soient occupés en même temps. De toute façon, le transfert direct, sans passer par un fichier, n'a volontairement pas été envisagé.

L'alimentation du lecteur (+5 V) sera prélevée sur la carte du programmeur, après l'interrupteur, qui servira ainsi dans les deux modes d'exploitation de l'appareil, selon les ordres du logiciel. Il suffit pour cela de relier les points « X » des deux cartes par un fil de câblage.

Quant à la « mise en boîte », elle pourra se faire dans tout coffret plastique ou métal offrant une surface de façade suffisante pour accueillir les deux cartes côte à côte. La forme « pupitre » semble de très loin la plus adaptée. Il est commode de coller, au ruban double face, un rectangle de mousse antistatique dans un coin de la façade : ainsi pourra-t-on y piquer les EPROMS attendant de prendre place sur un support, ce qui est une précaution élémentaire trop souvent négligée.

Les logiciels

Comme on l'imagine facilement, le pilotage de ce lecteur exige un logiciel capable de gérer la transmission série qu'il nous a fallu mettre en œuvre.

Le programme « LECROM » est, là encore, présenté en deux versions : BASIC à la figure 5.14, Turbo-Pascal à la figure 5.15.

Plus encore qu'en mode programmation, l'avantage de rapidité du Turbo-Pascal est évident à l'usage.

Figure 5.14.
Version BASIC du
logiciel LECROM.

```

10 REM ---- LECROM.BAS ----
20 CLS
30 PRINT"taille EPROM à lire (en k-octets) ?"
40 INPUT K
50 K=(1024*K)-1
60 CLS:PRINT"nom à donner au fichier?"
70 INPUT F$
80 OPEN"o",#1,F$+"*.ROM"
90 CLS:PRINT"ATTENTION, LE +5V DOIT ETRE COUPE"
100 OUT 888,64
110 PRINT"connecter l'EPROM à lire, puis presser ENTER"
120 INPUT Z$:CLS
130 PRINT"appliquer le +5V, puis presser ENTER"
140 INPUT Z$:CLS
150 PRINT"----- LECTURE EN COURS -----"
160 FOR G=0 TO K
170 D=0
180 FOR F=0 TO 7
190 OUT 888,F
200 B=INP(889)
210 IF (B AND 64)=64 THEN D=D+(2^F)
220 NEXT F
230 OUT 888,32
240 PRINT#1,D;
250 NEXT G
260 CLS:PRINT"----- COUPER LE +5V -----"
270 BEEP:END
280 REM (c)1991 Patrick GUEULLE

```

```

program LECROM;
uses crt;
var k:integer;
g:integer;
f:integer;
b:integer;
d:real;
h:string[12];
c:boolean;
a:text;
x:integer;
begin
clrscr;
writeln('TAILLE MEMOIRE A LIRE (en k-octets) ?');
readln(k);

```

Figure 5.15.
Version Turbo-
Pascal du logiciel
LECROM.

Suite

Figure 5.15.

```

k:=(k*1024)-1;
clrscr;
writeln('NOM A DONNER AU FICHIER ?');
readln(h);
h:=h+'.rom';
assign(a,h);
rewrite(a);
clrscr;
writeln('ATTENTION, LE +5V DOIT ETRE COUPE !');
port[888]:=64;
writeln('CONNECTER MEMOIRE A LIRE, puis presser ENTER');
readln;
clrscr;
writeln('APPLIQUER LE +5V, puis presser ENTER');
readln;
clrscr;
writeln('----- LECTURE EN COURS -----');
writeln;
for g:=0 to k do
begin
d:=0;
for f:=0 to 7 do
begin
port[888]:=f;
b:=port[889];
if (b and 64)=64 then d:=d+(exp(f*ln(2)));
x:=round(d);
end;
port[888]:=32;
write(a,' ',x,' ');
end;
sound(440);
delay(1000);
nosound;
clrscr;
writeln('COUPER LE +5V');
writeln('puis retirer la mémoire');
close(a);
end.
(* COPYRIGHT 1991 Patrick GUEULLE *)

```

Insistons toutefois à nouveau sur le fait qu'on peut se permettre d'attendre quelques instants qu'une EPROM se trouve lue, même si on sait que quelques millisecondes suffiraient si elle était directement connectée aux bus de l'ordinateur. Un signal sonore est d'ailleurs prévu en fin d'opération, ce qui permet éventuellement de vaquer à d'autres occupations en attendant...

Le programme « VEROM » de la figure 5.16 (BASIC) ou de la figure 5.17 (Turbo-Pascal) permet pour sa part de vérifier la conformité d'une EPROM avec un fichier de référence dont on spécifie le nom (sans l'extension « . ROM », automatiquement ajoutée par le logiciel). Cette vérification par rapport à la disquette permet de contrôler le succès d'une opération de programmation, et aussi de repérer toute dégradation d'une EPROM programmée depuis longtemps ou ayant subi des mésaventures.

```

10 REM ---- VEROM.BAS ----
20 CLS
30 PRINT"nom du fichier de référence ?"
40 INPUT F$
50 OPEN"i",#1,F$+".ROM"
60 PRINT"ATTENTION, LE +5V DOIT ETRE COUPE"
70 OUT 888,64
80 PRINT"connecter l'EPROM à lire, puis presser ENTER"
90 INPUT Z$
100 PRINT"appliquer le +5V, puis presser ENTER"
110 INPUT Z$
120 PRINT"----- COMPARAISON EN COURS -----"
130 G=0
140 D=0
150 FOR F=0 TO 7
160 OUT 888,F
170 B=INP(889)
180 IF (B AND 64)=64 THEN D=D+(2^F)
190 NEXT F
200 OUT 888,32
210 INPUT#1,C
220 IF EOF(1) THEN 260
230 IF C<>D THEN PRINT G,D,"au lieu de",C
240 G=G+1
250 GOTO 140
260 PRINT"----- COUPER LE +5V -----"
270 BEEP:END
280 REM (c)1991 Patrick GUEULLE

```

Figure 5.16.

Version BASIC du logiciel VEROM.

```

10 REM ---- VIROM.BAS ----
20 CLS
30 PRINT"taille de l'EPROM (en k-octets) ?"
40 INPUT K:CLS
50 K=(K*1024)-1
60 PRINT"ATTENTION, LE +5V DOIT ETRE COUPE"
70 OUT 888,64
80 PRINT"connecter l'EPROM à tester, puis presser ENTER"
90 INPUT Z$:CLS
100 PRINT"appliquer le +5V, puis presser ENTER"
110 INPUT Z$
120 PRINT"----- TEST DE VIRGINITE EN COURS -----"
130 FOR G=0 TO K
140 FOR F=0 TO 7
150 OUT 888,F
160 B=INP(889)
170 IF (B AND 64)=64 THEN 190
180 PRINT"*";:F=7
190 NEXT F
200 OUT 888,32
210 NEXT G
220 PRINT"----- COUPER LE +5V -----"
230 BEEP:END
240 REM (c)1991 Patrick GUEULLE

```

Figure 5.17.

Version BASIC du logiciel VIROM.

Figure 5.18.
Version Turbo-
Pascal du logiciel
VEROM

```

program VEROM;
uses crt;
var h:string[12];
    g:integer;
    d:real;
    f:integer;
    b:integer;
    c:integer;
    a:text;
    x:integer;
    e:boolean;
begin
  clrscr;
  writeln('NOM DU FICHIER DE REFERENCE ?');
  readln(h);
  h:=h+'.rom';
  assign(a,h);
  reset(a);
  clrscr;
  writeln('ATTENTION! LE +5V DOIT ETRE COUPE');
  port[888]:=64;
  writeln('CONNECTER MEMOIRE A LIRE, puis presser ENTER');
  readln;
  clrscr;
  writeln('APPLIQUER LE +5V, puis presser ENTER');
  readln;
  clrscr;
  writeln('----- COMPARAISON EN COURS -----');
  g:=0;
  repeat
    d:=0;
    for f:=0 to 7 do
      begin
        port[888]:=f;
        b:=port[889];
        if (b and 64)=64 then d:=d+(exp(f*ln(2)));
        x:=round(d);
      end;
    port[888]:=32;
    read(a,c);
    e:=seekeof(a);
    if c<>d then writeln(g,' : ',x,' au lieu de ',c);
    g:=g+1;
  until e=true;
  writeln;
  sound(440);
  delay(1000);
  nosound;

```

```

writeln('COUPER LE +5V');
close(a);
end.
(* COPYRIGHT 1991 Patrick GUEULLE *)

```

```

program VIROM;
uses crt;
var k:integer;
    g:integer;
    f:integer;
    b:integer;
begin
  clrscr;
  writeln('taille EPROM en k-octets ?');
  readln(k);
  k:=(k*1024)-1;
  clrscr;
  writeln('ATTENTION, LE +5V DOIT ETRE COUPE');
  port[888]:=64;
  writeln('Connecter EPROM à tester, puis presser ENTER');
  readln;
  clrscr;
  writeln('Appliquer le +5V, puis presser ENTER');
  readln;
  clrscr;
  writeln('----- TEST DE VIRGINITE EN COURS -----');
  for g:=0 to k do
    begin
      for f:=0 to 7 do
        begin
          port[888]:=f;
          b:=port[889];
          if (b and 64)<>64 then write('*');
        end;
        port[888]:=32;
      end;
      writeln('----- COUPER LE +5V -----');
    end;
  end.
  (* COPYRIGHT 1991 Patrick GUEULLE *)

```

Figure 5.19.
Version Turbo-
Pascal du logiciel
VIROM.

VEROM n'affiche que le contenu des adresses défectueuses, tandis que VIROM affiche simplement des astérisques dont le nombre est fonction de la gravité du défaut de virginité. Cela leur permet d'être plus rapides que LECROM ou PROGROM.

LA PROGRAMMATION ET LA LECTURE DES RAM ZEROPOWER

Les RAM à pile lithium incorporée ou ZEROPOWER remplacent avantageusement les EPROM lors des opérations de développement et de mise au point.

L'effacement aux UV n'est plus nécessaire, tandis que la programmation s'effectue en un clin d'œil.


```

port(888):=0;
clrscr;
writeln('COUPER LE +5V puis presser ENTER');
readln;
clrscr;
writeln('RETIRER LA RAM A PILE');
end.
(* COPYRIGHT 1991 Patrick GUEULLE *)

```

Rien n'interdit cependant de renommer « PROGRAM.EXE » le fichier exécutable obtenu à l'issue de la compilation, ce qui a d'ailleurs été fait sur la disquette jointe à l'ouvrage.

UNE ALIMENTATION DE SÉCURITÉ POUR PROGRAMMATION D'EPROM

La programmation de mémoires EPROM fait appel à des tensions d'alimentation bien spécifiques : du + 5 V évidemment (le V_{cc} ou V_{dd} général), mais aussi un V_{pp} dont la valeur diffère d'un type de mémoire à l'autre.

Autrefois, la plupart des EPROM se programmaient en présence d'un V_{pp} de + 25 V. Les progrès de la technique aidant sont apparues des versions se contentant de + 21 V ou même de + 12,5 V, parfois sans le moindre changement de référence ! Or, il faut savoir que le V_{pp} préconisé doit être respecté au demi-volt près, sous peine de risquer au mieux de rater la programmation, au pire de « tuer » l'EPROM, mais en tout cas d'abréger sa durée de vie.

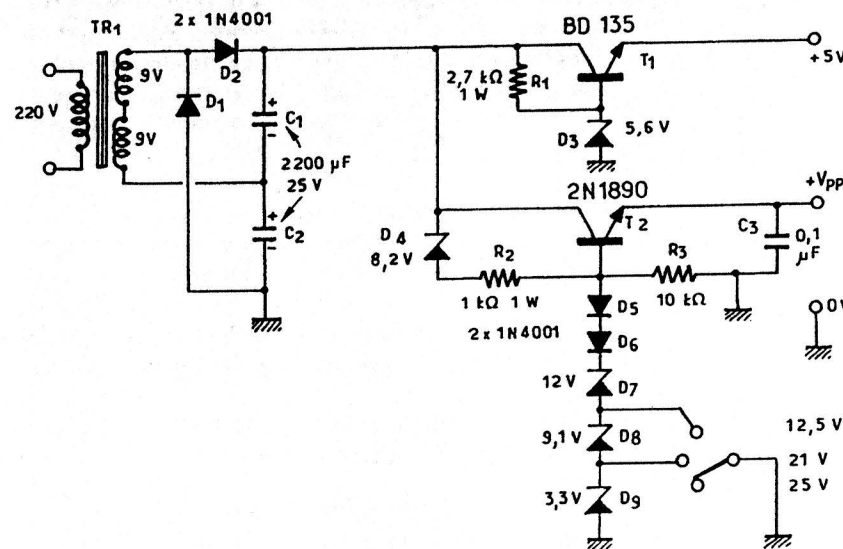
Parallèlement, le V_{pp} ne doit en aucun cas être appliqué à la mémoire en l'absence du + 5 V, même pendant une fraction de seconde : il doit être appliqué après et retiré avant l'alimentation principale, sous peine de destruction pure et simple de l'EPROM.

Nous avons choisi de faire confiance à nos lecteurs en ce qui concerne le strict respect des instructions données par le logiciel de programmation, mais une coupure de courant subite est toujours possible. Il est donc souhaitable qu'une sécurité fasse en sorte que le V_{pp} s'écroule plus vite que le V_{cc} , et mette plus de temps à se rétablir lorsque le secteur reviendra. Ce cycle devra par ailleurs être respecté quelles que soient les consommations respectives sur le V_{cc} et sur le V_{pp} (des différences sensibles existent entre EPROM selon le type et même selon la marque).

Le schéma de la figure 5.23 respecte le cahier des charges qui vient d'être défini, en faisant appel à un strict minimum de composants. Il serait évidemment possible de faire encore

mieux avec, par exemple, des régulateurs intégrés asservis l'un à l'autre, mais cela ne s'impose nullement.

Figure 5.23.
Schéma de
l'alimentation.



Un redresseur du type doubleur de tension permet d'obtenir à partir d'un simple transformateur 2×9 V (ou 18 V) la tension non régulée nécessaire, dont la valeur est assez élevée (une cinquantaine de volts). Cette configuration permet par ailleurs de se contenter de condensateurs de filtrage de 2 200 µF 25 V, pas trop encombrants.

Un premier transistor ballast BD 135 polarisé par une zener de 5,6 V fournit le + 5 V nécessaire sous un courant maximum d'environ 100 mA. Devant chuter près de 45 V, ce transistor chauffera assez fortement : on ne négligera pas la question de son refroidissement, quitte à utiliser un radiateur plus important que d'habitude.

Mais pourquoi infliger une telle dissipation à ce transistor ? Tout simplement parce que ce régulateur dispose ainsi d'une très forte réserve d'énergie dans les condensateurs de filtrage. En cas de coupure secteur, le + 5 V ne commencera à chuter que lorsque la tension non régulée sera tombée aux environs de 7 V. Pendant ce temps, V_{pp} aura déjà fortement diminué !

Cette première ligne de défense est doublée par une seconde sécurité : une zener de 8,2 V en série dans la branche supérieure du pont de polarisation du ballast du V_{pp} (2N 1890 ou similaire, tenant au moins 50 V). Grâce à la 10 k Ω qui « rappelle » la base du ballast à la masse, ce dernier se bloquera au plus tard lorsque la tension non régulée sera tombée à 8,2 V, et probablement un peu avant. Mais nous savons qu'à cet instant, le + 5 V est toujours fidèle au poste ! Inversement, à la mise en route ou au retour du secteur après une coupure, le V_{pp} n'apparaîtra qu'une fois le + 5 V solidement établi.

La sélection de la valeur de V_{pp} parmi les trois « grands classiques » s'opère par court-circuit partiel d'un pont de diodes zener et de diodes silicium calculé de façon à approcher d'aussi près que possible les valeurs recommandées, et à dériver aussi peu que possible en température (rappelons en effet que les dérives des zeners et des diodes ordinaires sont opposées).

Bien évidemment, d'autres combinaisons pourraient être retenues selon les besoins exacts de chacun.

Il nous semble que la simple manœuvre d'un commutateur est à la fois plus confortable et plus sûre que le réglage d'un potentiomètre en surveillant un voltmètre, mais encore faut-il ne pas se tromper de position ! Bien souvent, les EPROM exigeant un V_{pp} différent de 25 V portent un marquage particulier, mais on ne saurait trop recommander de consulter le catalogue du fabricant, notamment pour ce qui est des versions les plus récentes (notamment CMOS).

Le circuit imprimé de la figure 5.24 supporte tous les composants du montage, transformateur secteur compris : les pastilles qui lui sont affectées devront peut-être être déplacées selon le brochage exact de la pièce disponible, à moins qu'on ne préfère le monter à part.

Le 2N 1890 devra, comme le BD 135, être refroidi énergiquement bien qu'il chauffe nettement moins. Pour un V_{pp} de 12,5 V, il doit malgré tout chuter plus de 38 V ce qui, sous 80 mA, correspond à près de 3 W ! En 25 V, par contre, on peut « tirer » jusqu'à 110 mA sans excéder ce chiffre qui correspond à la dissipation maximale du 2N 1890.

Si des courants supérieurs étaient nécessaires (par exemple pour programmer simultanément plusieurs mémoires connectées en parallèle), il serait facile de trouver des transistors sup-

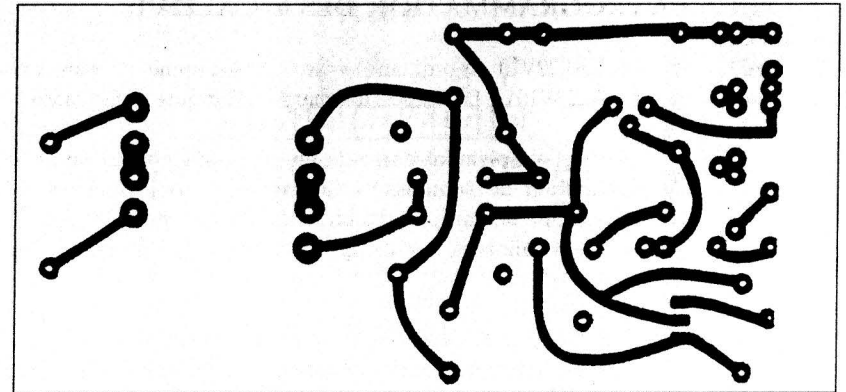


Figure 5.24. Tracé du circuit imprimé.

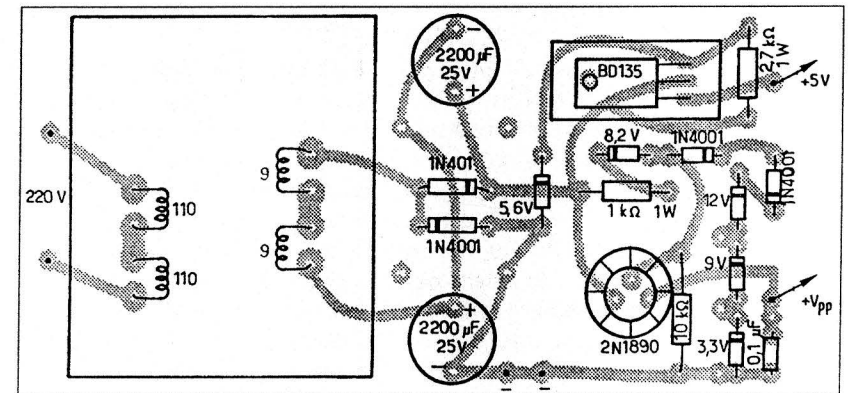


Figure 5.25. Cablage de l'alimentation.

Liste des composants de l'alimentation

- 1 $\times$ BD 135.
- 1 $\times$ 2N 1890.
- 4 $\times$ 1N 4001.
- 1 $\times$ zener 12 V.
- 1 $\times$ zener 9,1 V.
- 1 $\times$ zener 3,3 V.
- 1 $\times$ zener 8,2 V.
- 1 $\times$ zener 5,6 V.
- 1 $\times$ 1 k Ω 1 W.
- 1 $\times$ 2,7 k Ω 1 W.
- 1 $\times$ 10 k Ω .
- 1 $\times$ 0,1 μ F.
- 2 $\times$ 2 200 μ F 25 V.
- 1 transformateur 220-18 V ou 220-2 $\times$ 9 V.
- 1 cordon secteur.
- 1 commutateur 3 positions.

portant davantage de courant mais il faudrait veiller à ce que leur gain reste convenable (on choisirait de préférence des darlington).

Bien entendu, il est essentiel que le transformateur soit capable de fournir le courant nécessaire. En version de base, un modèle de 8 à 10 VA suffit largement. Une fois câblé, ce montage pourra être intégré dans le boîtier du programmeur où beaucoup de place reste certainement disponible, ou dans un coffret séparé. Dans tous les cas, et surtout dans le premier, on soignera l'isolation de la partie 220 V et on intercalera éventuellement un interrupteur et un fusible, quitte à ajouter un voyant de contrôle.

LA PROGRAMMATION DES ispGAL22V10

Un GAL22V10 « ordinaire » se « programme », mais un ispGAL22V10 de Lattice se « télécharge ». Pourquoi cette nuance ?

Cela tout simplement parce qu'un processus normal de programmation de composants fait appel, nous l'avons vu, à l'application de tensions inhabituelles, pendant des durées parfaitement calibrées, sur un grand nombre de broches tour à tour.

À l'inverse, un téléchargement consiste tout simplement en un dialogue série entre le composant et le PC, sur un nombre très limité de fils et sous une tension de 5 volts. Tout le processus délicat se déroule alors automatiquement, à l'intérieur même du composant, de façon complètement « transparente » pour l'opérateur.

Alors que les programmeurs classiques agissent sur les broches « normales » du composant, le téléchargement peut se faire par l'intermédiaire de broches exclusivement réservées à cet usage. Il devient donc possible de programmer ou reprogrammer le composant à l'intérieur même de son circuit utilisateur, celui-ci restant d'ailleurs sous tension ! Lattice appelle cela *In Situ Programmability*, d'où la dénomination « isp » de cette famille de composants. Mais contrairement aux composants téléchargeables basés sur une mémoire RAM, l'ispGAL22V10 conserve sa programmation jusqu'à ce qu'on l'efface volontairement. Une fois programmé, c'est un 22V10 comme les autres, qui peut, si nécessaire, être débrogé ou changé de carte sans perdre le « schéma » qu'il contient. C'est particulièrement intéressant pour le développement de prototypes...

Mais comment ajouter des broches de téléchargement à un 22V10 tout en maintenant une compatibilité « pin pour pin » avec les modèles ordinaires ? Eh bien tout simplement en se limitant à la présentation en boîtier PLCC à 28 broches, dans laquelle quatre contacts sont précisément inutilisés !

Cela conduit au brochage de la figure 5.26, sur lequel apparaissent les lignes de téléchargement SDO, SDI, MODE, et SCLK. Bien entendu, cette astuce serait inapplicable au boîtier DIP à 24 broches dans lequel on conditionne aussi les GAL22V10, mais il n'est pas interdit d'utiliser un adaptateur.

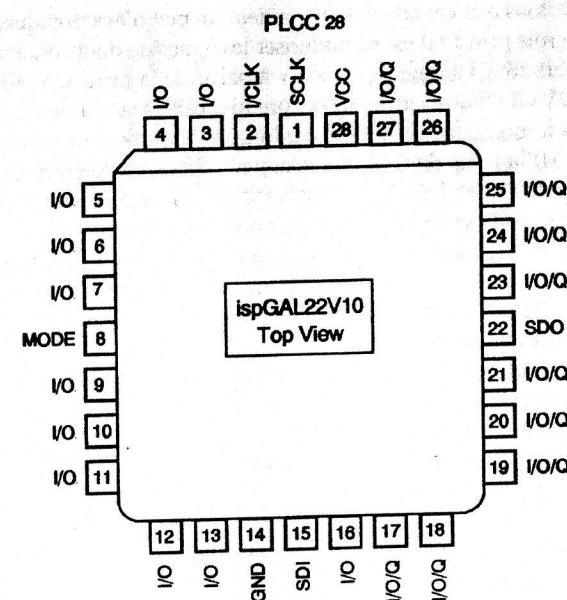


Figure 5.26.
Brochage de
l'ispGAL22V10
(boîtier PLCC 28).

Nous allons étudier successivement deux approches permettant de programmer des ispGAL22V10 à l'aide d'un compatible PC :

- l'utilisation pure et simple du *starter kit* vendu par Lattice, qu'il faut de toute façon acquérir si on souhaite étendre l'expérience aux composants de la famille ispLSI ;
- la réalisation d'un programmeur « maison » utilisable avec le logiciel offert par Lattice sur la disquette jointe à cet ouvrage.

L'équipement nécessaire à la mise en œuvre de la première solution est particulièrement simple : un câble de raccordement et du logiciel.

Bien que le data book Lattice contienne tout le détail du protocole de téléchargement à partir d'un fichier JEDEC issu du premier compilateur venu (par exemple Prologic), il n'est vraiment intéressant d'écrire ses propres logiciels que dans le cas où c'est le système hôte lui-même qui doit pouvoir reprogrammer le GAL (une bibliothèque en langage C est disponible pour faciliter ce travail de programmation). On utilisera donc en général les utilitaires Windows développés à cet effet par Lattice, et fournis dans le starter kit.

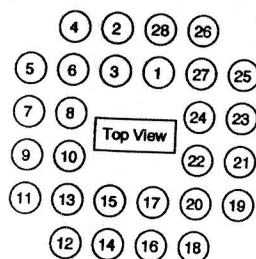
Sur le plan matériel, les cordons de téléchargement fournis dans le starter kit sont munis d'un jack modulaire à huit contacts (RJ-45), qu'il faut raccorder au port LPT1: du PC par un adaptateur DB25 fourni.

Notons que cet adaptateur contient un peu d'électronique, dont le rôle principal est de bufferiser les lignes de données, et qu'on doit donc l'alimenter en + 5 V à partir de la carte accueillant le 22V10. L'autre extrémité du premier câble est munie du même jack modulaire, le second cordon étant équipé d'un connecteur AMP au pas de 2,54 mm. Dans les deux cas, le raccordement au 22V10 est laissé aux bons soins de l'utilisateur, qui est censé prévoir la prise *ad hoc* sur sa carte d'application.

Rien n'est prévu dans cette connectique, dont la figure 5.27 résume les caractéristiques, pour la programmation hors-carte de 22V10 à des fins de développement. Mais nous allons y remédier de ce pas !

Le problème posé consiste à relier à un support PLCC28 dont le brochage correspond au plan de la figure 5.28, d'une part six des broches de la fiche AMP du cordon de téléchargement, et d'autre part une alimentation 5 V.

Figure 5.28.
Brochage des
supports pour
PLCC28.



C'est dans ce but qu'a été dessiné le petit circuit imprimé de la figure 5.29, qu'il faut équiper selon la figure 5.30.

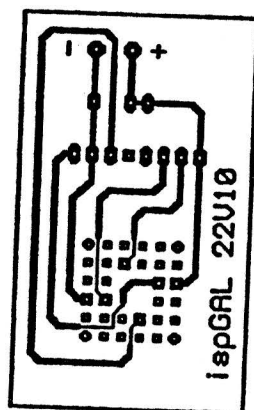


Figure 5.29.
Circuit imprimé de
l'adaptateur PLCC28.

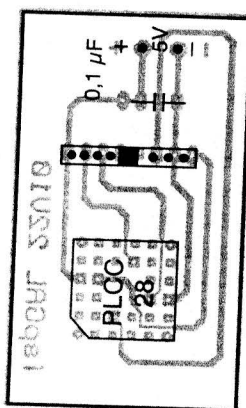


Figure 5.30.
Plan de câblage de
l'adaptateur PLCC28.

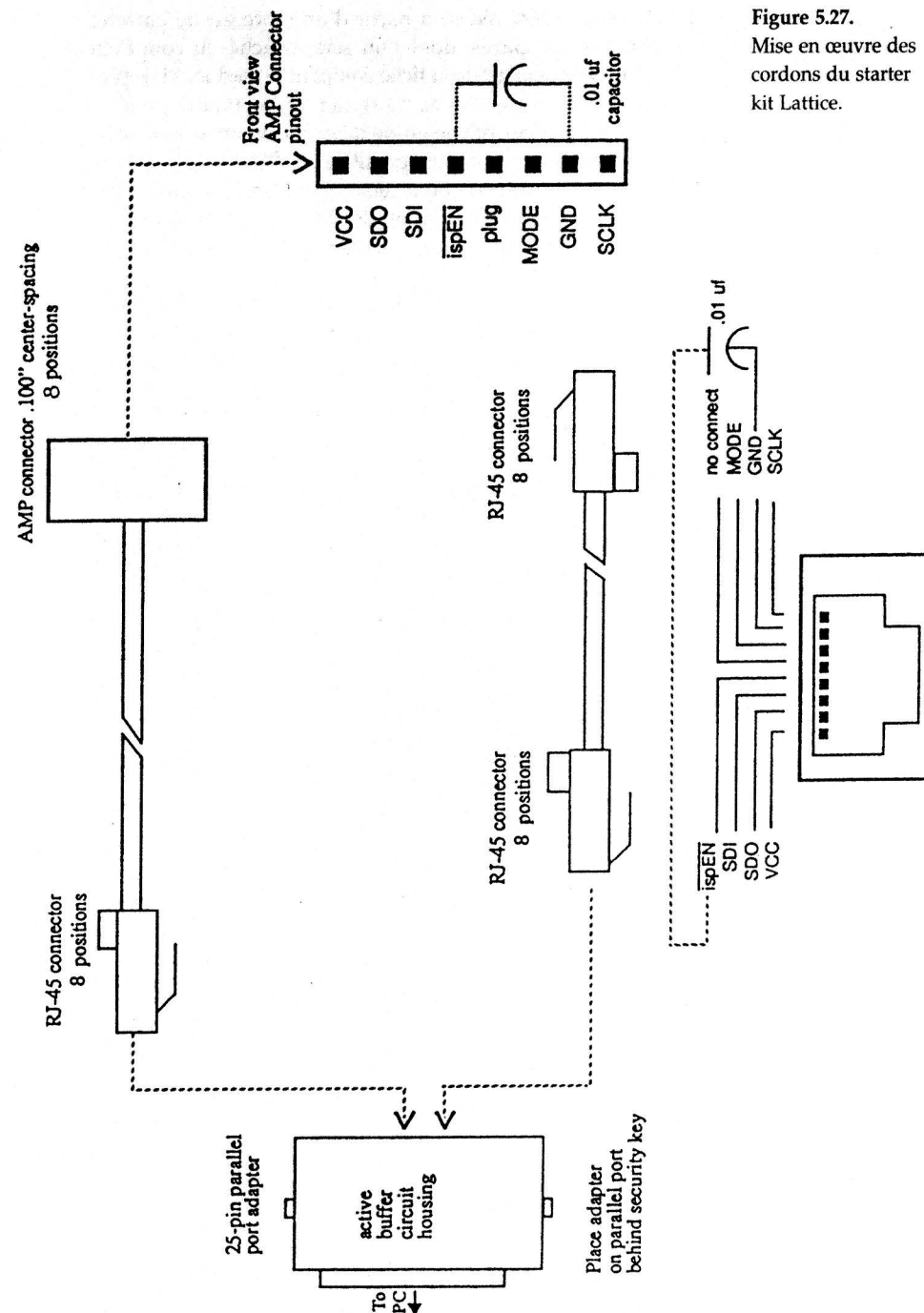
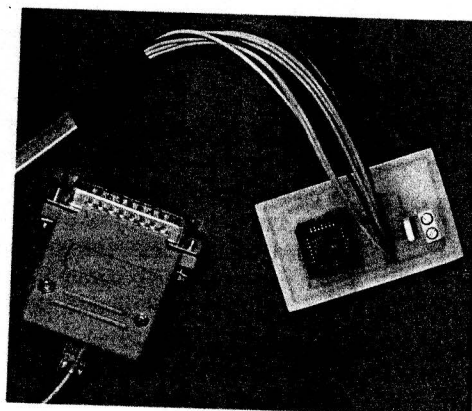
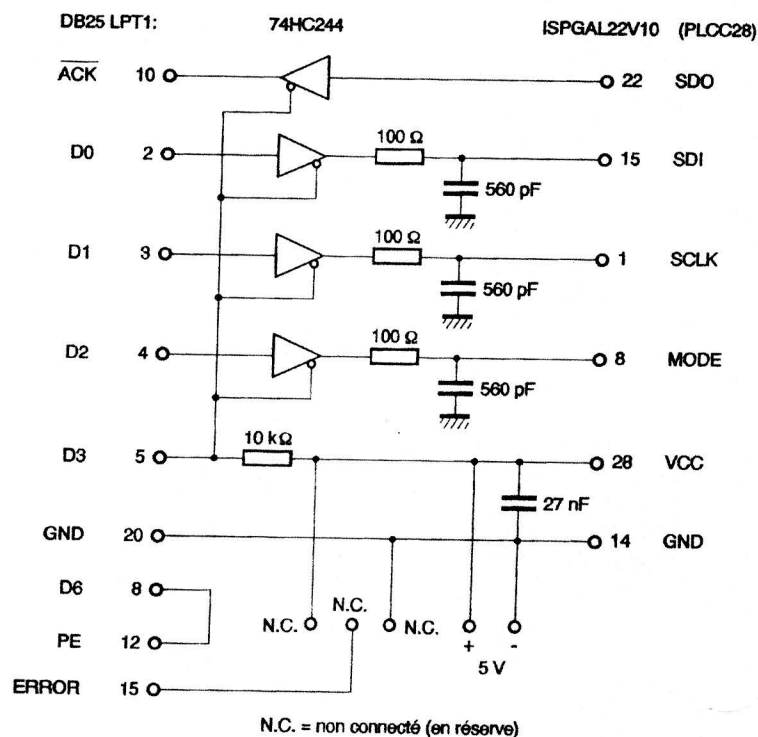


Figure 5.27.
Mise en œuvre des
cordons du starter
kit Lattice.

Le connecteur sera réalisé à partir d'un morceau de barrette sécable à picots carrés, dont l'un sera arraché ou coupé en regard du détrompeur de la fiche équipant le cordon. Mais plutôt que d'utiliser les cordons du starter kit, on pourra préférer construire de toutes pièces un petit programmeur se raccordant directement sur le port parallèle du PC. Ce sera notamment une bonne solution pour ceux de nos lecteurs qui souhaiteraient se limiter à l'utilisation des logiciels fournis avec cet ouvrage, sans investissement supplémentaire.

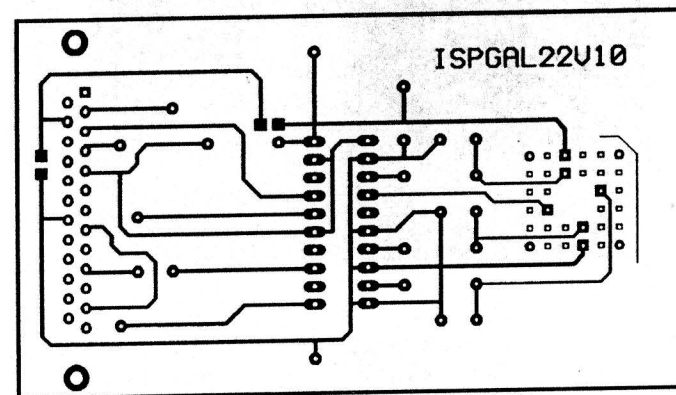
Le schéma de la figure 5.31 est évidemment inspiré de celui préconisé par Lattice, mais sa réalisation fait appel à un circuit intégré plus courant.

Figure 5.31.
Schéma du
programmeur
d'ispGAL22V10.



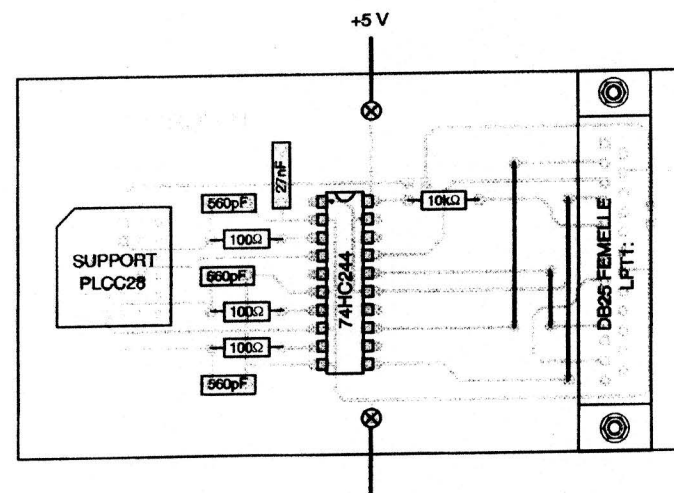
Le circuit imprimé de la figure 5.32 réunit l'ensemble du matériel nécessaire, ce qui réduit au strict minimum la longueur des connexions.

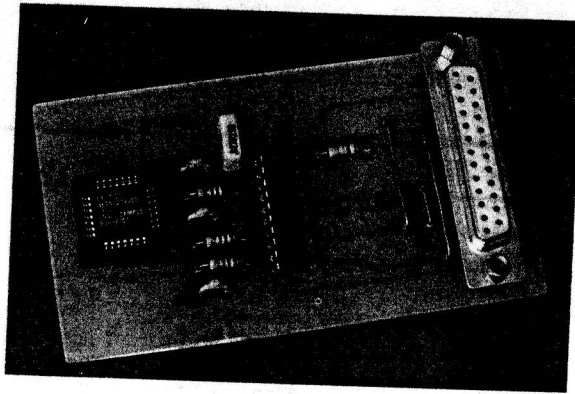
Figure 5.32.
Tracé du circuit
imprimé du pro-
grammeur



Après câblage selon la figure 5.33 (sans oublier les trois straps !), il suffira de relier la carte à une alimentation 5 volts (par exemple la broche n°1 d'une prise DB15 joystick du PC), et à un très court cordon DB25 double mâle (dit changeur de genre) assurant une liaison souple avec le port parallèle LPT1: du même PC. Bien entendu, l'exploitation de ce programmeur est strictement identique à celle du cordon de téléchargement.

Figure 5.33.
Plan de câblage du
programmeur.





Dans les deux cas, il est très important de bien noter la différence de numérotation des broches entre les 22V10 en boîtiers DIP24 et PLCC28, dont le tableau de la figure 5.34 donne le détail (abstraction faite des quatre broches de téléchargement signalées à la figure 5.26). En effet, les noms de signaux qu'il faut utiliser avec Prologic sont liés aux numéros des broches du boîtier DIP ! Lorsqu'on écrira le code source d'une application destinée à un ispGAL22V10 en boîtier PLCC, il faudra veiller à éviter toute confusion.

Figure 5.34.
Correspondance
de brochage entre
boîtiers DIP24 et
PLCC28.

BROCHE DIP 24	BROCHE PLCC 28	FONCTION
1	2	entrée / horloge
2	3	entrée
3	4	entrée
4	5	entrée
5	6	entrée
6	7	entrée
7	9	entrée
8	10	entrée
9	11	entrée
10	12	entrée
11	13	entrée
12	14	GND
13	16	entrée
14	17	entrée / sortie
15	18	entrée / sortie
16	19	entrée / sortie
17	20	entrée / sortie
18	21	entrée / sortie
19	23	entrée / sortie
20	24	entrée / sortie
21	25	entrée / sortie
22	26	entrée / sortie
23	27	entrée / sortie
24	28	VCC

LA PROGRAMMATION DES ispLSI 1016 ET 2032

Le starter kit Lattice, avec son logiciel de développement de code « PDS », étant indispensable à la bonne mise en œuvre des ispLSI 1016 et 2032, la programmation (ou la reprogrammation après effacement) de ces composants « sur la table » ne nécessite que la réalisation d'un petit adaptateur muni du support PLCC approprié. Ce sera bien évidemment d'une inestimable utilité en phase de développement ou de création de prototype.

La figure 5.35 fournit le tracé du circuit imprimé simple face nécessaire, et la figure 5.36 le plan de câblage correspondant.

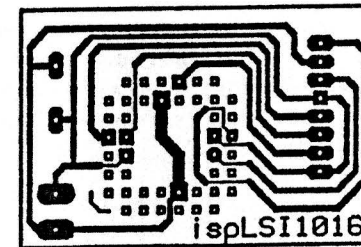


Figure 5.35.
Circuit imprimé
de l'adaptateur
PLCC44.

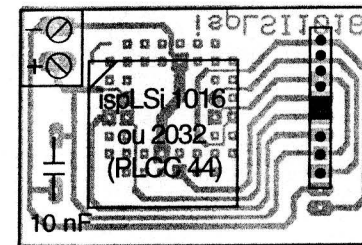
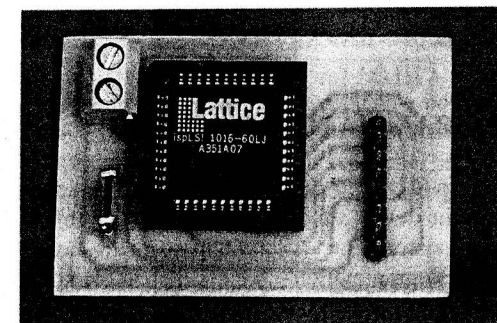


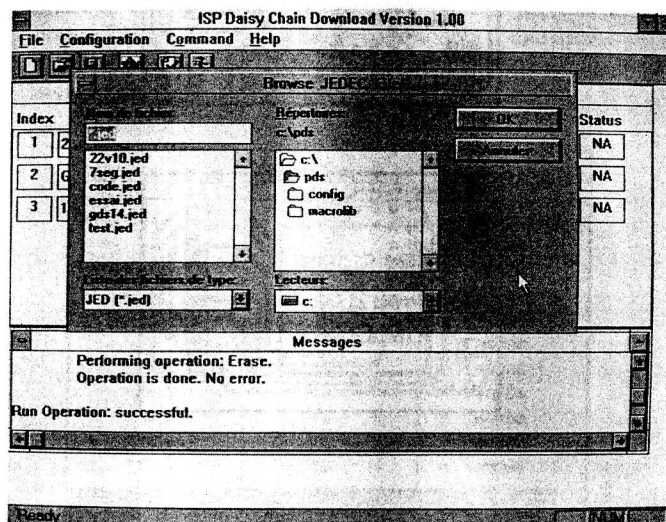
Figure 5.36.
Plan de câblage
de l'adaptateur
PLCC44.



On veillera à bien respecter l'orientation du support PLCC44 (du type à picots et non CMS), repérée par un coin coupé, et à créer un détrompeur en supprimant la broche correspondante de la barrette sécable à picots carrés qui sert de connecteur mâle.

Pour utiliser ce petit accessoire (compatible avec les ispLSI1016 et 2032), il suffit d'y brancher le cordon de téléchargement du starter kit et (seulement après insertion du composant !) une alimentation 5 V, que l'on peut tout simplement prélever sur la broche n°1 d'une prise joystick du PC.

Le reste est purement une affaire de logiciel : programmation, vérification et quand il le faut effacement, se font par simple appel de l'option correspondante dans un menu de PDS, ou bien à l'aide du logiciel qui, fourni sur la disquette d'accompagnement de cet ouvrage, sert déjà à programmer les ispGAL22V10 sous Windows.



UN PROGRAMMATEUR DE PIC

Même dans le cadre d'un *starter kit* Microchip, acheter un programmeur de PIC du commerce revient forcément plus cher que d'en construire un ! À condition de mettre à contribution le mode de programmation « série » disponible sur les PIC les plus récents (et notamment sur le très apprécié 16C84), il est

possible de réaliser un programmeur étonnamment simple et entièrement compatible avec les puissants logiciels que nous avons dénichés pour vous sur Internet.

Le saviez vous ? Plusieurs références de la famille PIC appliquent un concept voisin de celui des composants « isp », qui permet de les programmer (ou reprogrammer) directement sur la carte où ils se trouvent, sans même couper l'alimentation.

Le passage en mode « programmation » se fait en portant la broche de reset (/MCLR) à un « supervoltage » compris pour la circonstance entre 12 et 14 volts. Dès lors, la broche RB6 devient une entrée d'horloge (CLOCK), et la broche RB7 une entrée-sortie de données (DATA).

V_{dd} et V_{ss} conservent leur affectation normale, à savoir l'alimentation du PIC sous sa tension nominale de 5 volts, tandis que toutes les autres broches deviennent inopérantes. Il est alors possible de programmer, lire, ou même effacer le PIC rien qu'en dialoguant avec lui, en série, sur la ligne DATA.

Les règles de ce dialogue sont entièrement dévoilées dans un document d'une dizaine de pages (PIC16C84 EEPROM *Memory Programming Specification*) disponible auprès de Microchip ou consultable sur le CDROM diffusé par ce fabricant. Il est toutefois possible d'assouplir très sérieusement ces règles assez contraignantes, à condition de se limiter à des applications de prototypage ou de développement.

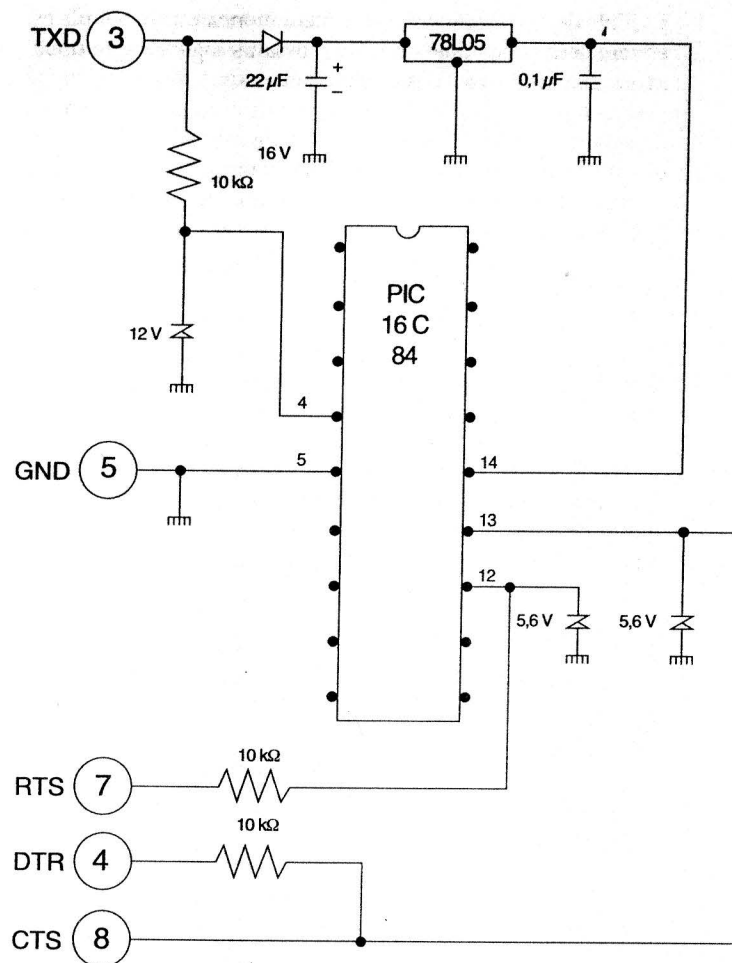
L'idée de base à partir de laquelle a été conçu le schéma, extrêmement simple, de la figure 5.37, est d'alimenter le PIC par une tension prélevée sur la ligne TXD d'un port série (RS 232), et d'utiliser les autres lignes de celui-ci pour le dialogue.

Cela évite tout recours à une alimentation extérieure ou même à une quelconque pile, et se révèle extrêmement pratique à l'usage.

Si cette disposition ne pose pas de problème pour l'obtention d'un « bon » 5 volts à l'aide d'un régulateur 78L05, la récupération d'un minimum de 12 volts pour la broche/MCLR suppose par contre que le port série soit un authentique RS 232 C délivrant des niveaux d'au moins + et -12 volts.

En aucun cas le programmeur ne pourra donc fonctionner sur les PC équipés de ports série travaillant en 5 volts. On se

Figure 5.37.
Schéma du
programmeur de
PIC16C84.



méfiera en particulier des portables, qui font souvent appel à des artifices d'économie d'énergie pas toujours très orthodoxes.

En pareil cas, le remède consisterait à ajouter un port série de bonne qualité, par exemple sous la forme d'une carte « multi I/O », accessoire fort peu coûteux et bien utile par ailleurs.

Sachant maintenant que des tensions d'au moins 12 volts, aussi bien positives que négatives, sont réputées présentes sur les lignes du port série, il nous a fallu limiter celles-ci à des niveaux inoffensifs pour le PIC, d'où la présence de quelques diodes zener qui travaillent d'ailleurs dans les deux sens.

Après gravure d'un petit circuit imprimé conforme au tracé de la figure 5.38, ces différents composants seront montés selon le plan de la figure 5.39, en veillant à bien respecter l'orientation des éléments polarisés.

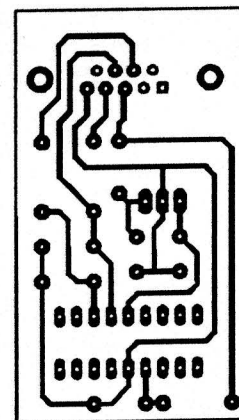


Figure 5.38.
Circuit imprimé du
programmeur de PIC.

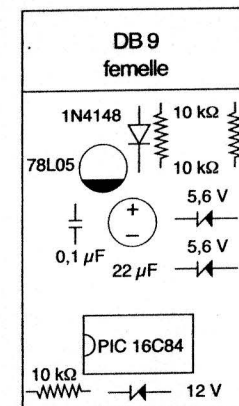


Figure 5.39.
Plan de câblage du
programmeur de PIC.

Liste des composants
du programmeur
d'EEPROM série

- 1 × 78L05.
- 1 × support 8 broches «tulipe».
- 1 × DB9 coudée femelle.
- 1 × 1N 4148.
- 2 × zener 4,7 V 1/4 W.
- 2 × 10 kW 1/4 W.
- 1 × 22 μF 16 V radial.
- 1 × 0,1 μF.

On ne tentera pas d'utiliser des zeners de tension inférieure à 5,6 volts, sous peine de non fonctionnement du montage. Le raccordement au port série du PC se fera soit par enfichage direct, soit au moyen d'un cordon DB9 mâle-femelle (connu sous le nom de rallonge de moniteur). Le cas échéant, on peut intercaler un adaptateur DB9-DB25, mais surtout pas un cordon croisé ou null-modem : un raccordement fil à fil s'impose !

Le logiciel de programmation

Le pilotage de ce petit programmeur nécessite un logiciel qui est offert au public, sur Internet, notamment à l'adresse suivante :

<http://www.sistudio.com>

Nous l'avons inclus sur la disquette d'accompagnement de cet ouvrage, à l'intention de nos lecteurs n'ayant pas encore accès au web, mais on ne saurait trop recommander le téléchargement de la toute dernière version.

Avant de lancer le logiciel principal (PIP-02), il est nécessaire d'installer un driver résident baptisé COM84. La syntaxe à employer est la suivante :

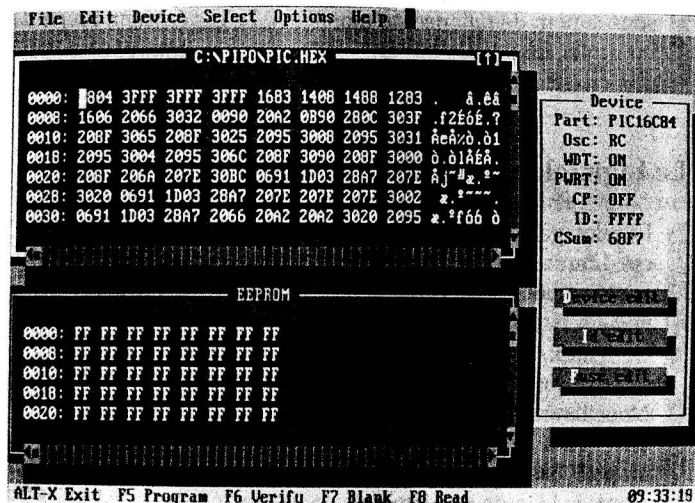
```
COM84 COMn (n étant le numéro du port série, de 1 à 4).
```

Il est commode d'automatiser les opérations en écrivant un petit fichier batch (nommé par exemple PIP.BAT), et dont le contenu pourrait être (dans le cas du port COM1):

```
COM84 COM1
PIP-02
COM84 REMOVE
```

La dernière ligne sert à désinstaller le driver lorsqu'on quitte le programme, afin d'éviter de possibles interférences avec d'autres logiciels.

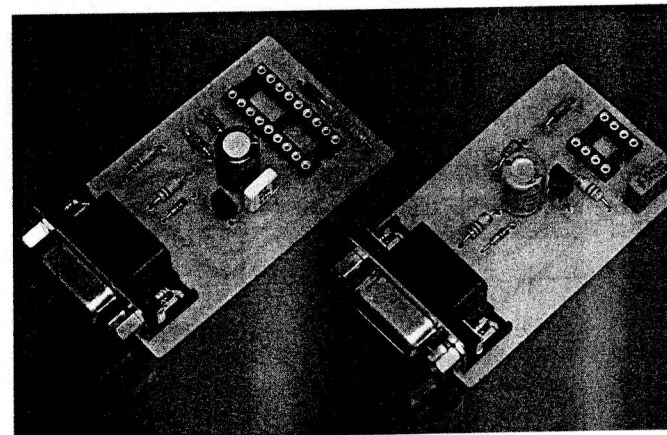
Rappelant à s'y méprendre la présentation des « vrais » outils Microchip, l'écran d'accueil de PIP-02 offre un bon confort d'utilisation. Plusieurs menus sont accessibles, avec lesquels il conviendra de se familiariser petit à petit. On n'oubliera pas, en particulier, de bien sélectionner le type de PIC à programmer (ou à lire) avant de lancer une quelconque opération. Dans l'état actuel du logiciel, seul le PIC16C84 peut être lu, programmé, et effacé, bien qu'il semble possible (sous toutes réserves) de lire (mais pas de programmer) le PIC16C71.



Bien entendu, on pensera à positionner correctement tous les « fusibles » avant de lancer une programmation. Ce devrait être une habitude déjà acquise par tous les adeptes des PIC...

Notons que le fichier servant à la programmation doit être disponible en format « INHX8M », qui est fort opportunément celui que génèrent la plupart des outils de développement pour PIC.

Un dernier détail qui a son importance : il est toujours possible d'effacer globalement le PIC, fusibles compris, par l'option *erase* du menu *device* !

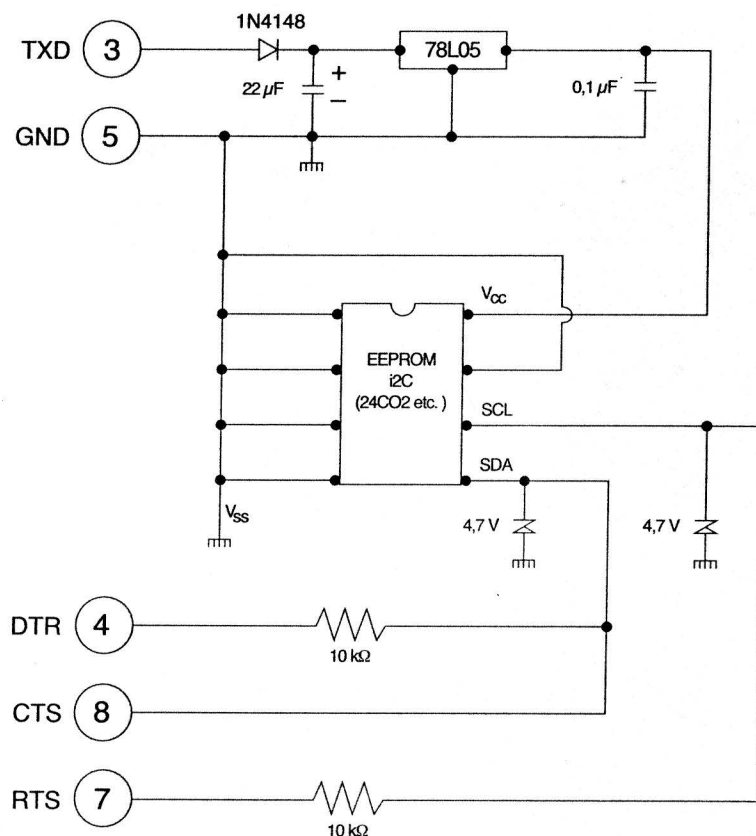


UN PROGRAMMATEUR DES EEPROM SÉRIE

Au fur et à mesure que se multiplient les applications des mémoires EEPROM série, on a de plus en plus souvent besoin de lire, recopier, ou modifier le contenu de ces composants courants et peu coûteux. Particulièrement simple, le petit montage de la figure 5.40 peut remplacer, à très peu de frais, les gros programmeurs « universels » qu'on utilise souvent pour ce genre de travail.

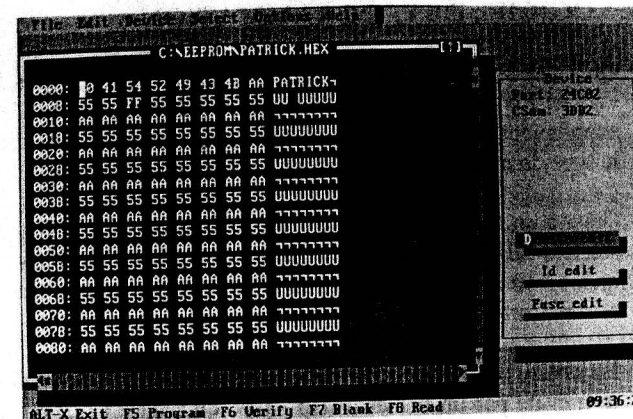
On remarquera que ce schéma n'est pas sans rappeler celui du montage précédent, et pour cause : il est, en effet, prévu pour être piloté par le même logiciel (PIP-02), dont le menu *select* propose de nombreuses références de mémoires EEPROM série compatibles avec le protocole « I2C » (principalement les 24CXX).

Figure 5.40.
Schéma du pro-
grammateur
d'EEPROM série.



Grâce au régulateur 78L05 et aux deux diodes zener de 4,7 V, aucune des broches de la mémoire ne pourra être portée à un potentiel négatif ou supérieur à 5 V, même en présence d'un port RS 232 travaillant, comme il se doit, en plus et moins 12 V. Pour des raisons de protection ESD, il sera cependant préférable, et d'ailleurs plus pratique, de n'enficher et désenficher

Rappelons qu'il s'agit d'une version provisoire, et que d'autres familles de mémoires seront probablement supportées ultérieurement. Leur place est même déjà réservée dans les menus ! Comme les PIC, les mémoires I2C communiquent avec le programmeur par une ligne de données (SDA) et une ligne d'horloge (SCL), sans nécessiter par contre aucune tension d'alimentation supérieure à 5 V. C'est dire que les restrictions formulées quant à la compatibilité du programmeur de PIC avec certains ports série de PC ne s'appliqueront pas ici.



© DUNOD - La photocopie non autorisée est un délit.

Figure 5.41. Circuit imprimé du
programmeur d'EEPROM série.

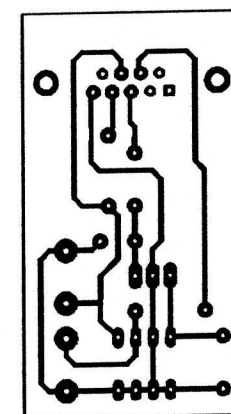
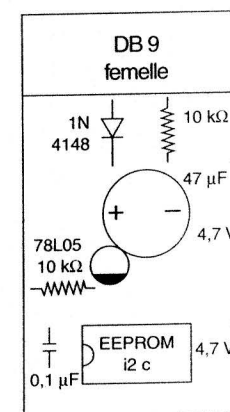


Figure 5.42. Plan de câblage du
programmeur d'EEPROM série.

les mémoires que lorsque le programmeur ne sera pas raccordé au PC. On branchera ainsi le montage sur le port série juste avant de lancer une opération de programmation ou de lecture, et on le débranchera juste après.

La réalisation de ce montage fait appel à un circuit imprimé épousant exactement les mêmes dimensions que celui du programmeur de PIC, et dont le tracé est reproduit à la figure 5.41.

Les deux montages seront donc parfaitement interchangeables, et sont d'ailleurs équipés du même modèle de prise DB9 coudée. L'implantation des quelques composants nécessaires, encore moins nombreux que sur le programmeur de PIC, se fera selon les indications de la figure 5.42.



Liste des composants
du programmeur
d'EEPROM série

- 1 × 78L05.
- 1 × support 8 broches «tulipe».
- 1 × DB9 coudée femelle.
- 1 × 1N 4148.
- 2 × zener 4,7 V 1/4 W.
- 2 × 10 kW 1/4 W.
- 1 × 22 µF 16 V radial.
- 1 × 0,1 µF.

LE LOGICIEL DE PROGRAMMATION

Qu'il s'agisse de travailler sur des PIC ou sur des mémoires, le logiciel PIP-02 s'installe et se lance de la même façon. C'est simplement par l'option *device* du menu *select* qu'on indiquera le type de mémoire qu'on souhaite lire ou programmer.

Grâce à un fichier d'initialisation qu'il met à jour lors de chaque session, PIP-02 se « souvient » toutefois du dernier composant qu'il a eu à traiter, et il conviendra donc de vérifier si sa dernière configuration correspond bien au travail à réaliser.

En lecture comme en programmation, le contenu de la mémoire transite par un fichier INTEL-HEX, qu'il n'est évidemment pas interdit de transcoder de ou vers un autre format, au moyen d'un utilitaire approprié.

Contrairement au cas des PIC, le logiciel ne sait pas effacer directement, en bloc, les mémoires EEPROM. On pourra contourner cette lacune (due tout bonnement au fait que ces mémoires ne sont pas technologiquement prévues pour cela !) en préparant à l'avance des fichiers HEX correspondant au contenu de mémoires vierges : soit par lecture de mémoires neuves, soit à l'aide de l'éditeur de fichiers incorporé à PIP-02.

L'effacement se fera alors très simplement par programmation du fichier correspondant à la référence de la mémoire dont on souhaite ainsi « écraser » le contenu.

1	Introduction aux composants programmables	9
2	Les mémoires programmables	13
3	Les réseaux logiques programmables	57
4	Les microcontrôleurs	95
5	Réalisation de programmeurs	111

6 LA DISQUETTE DU LIVRE

La disquette d'accompagnement de ce livre, qui ne peut en aucun cas être vendue séparément, est constituée de quatre parties détaillées ci-après :

PROLOGIC.EXE	282986
LECROM.PAS	970
PROGRAM.PAS	949
PROGROM.PAS	1554
VEROM.PAS	905
VIROM.PAS	613
LECROM.EXE	7520
PROGRAM.EXE	4048
PROGROM.EXE	6336
VEROM.EXE	7456
VIROM.EXE	2960
PROGROM.BAS	1254
TRANS.BAS	932
LECROM.BAS	655
VEROM.BAS	673
VIROM.BAS	606
PROGRAM.BAS	834
MQP <DIR>	
ISP <DIR>	
PIC <DIR>	

Son répertoire racine contient tous les programmes listés dans le livre, en versions GWBASIC et Turbo-Pascal.

À l'exception de TRANS.BAS qui n'est proposé qu'en BASIC, tous ces logiciels sont fournis sous la forme de deux fichiers source et d'un exécutable pouvant être directement lancé depuis le prompt du DOS (fichiers portant l'extension .EXE).

C'est en principe cette dernière forme, la plus performante, qu'il est recommandé d'employer, les deux autres étant surtout là pour permettre à nos lecteurs d'apporter des modifications personnelles aux programmes proposés.

Le fichier PROLOGIC.EXE contient, sous une forme compactée, l'équivalent des trois disquettes dont se compose le compilateur offert par Texas Instruments (© 1988, 1989, Inlab, Inc. PROLOGIC is a trademark of Inlab Incorporated).

Avant de pouvoir utiliser Prologic, il faut le décompacter dans un répertoire du disque dur ou sur une disquette de travail de 1,44 Mo.

On se placera tout d'abord dans le répertoire où l'on souhaite installer le logiciel, et on insérera la disquette du livre dans le lecteur A: ou B:.

Si celui-ci s'appelle A:, on tapera alors A: PROLOGIC pour lancer le processus au terme duquel le répertoire de travail contiendra la totalité des fichiers dont se compose PROLOGIC.

On se reportera alors au chapitre 3, et éventuellement à notre ouvrage *Logiciels PC pour l'électronique*, pour apprendre à utiliser correctement ce puissant logiciel.

Le sous-répertoire MQP de la disquette contient une version de démonstration parfaitement fonctionnelle du logiciel d'un programmeur d'EPROM britannique (© 1996, MQP Electronics Ltd.). Son principal intérêt est de comprendre un excellent éditeur de fichiers (PDED) offert sous une forme proche du shareware (lire attentivement le fichier READ.ME).

Avant de procéder à l'installation de ce logiciel, il faut recopier le contenu du sous-répertoire MQP sur une disquette intermédiaire. On placera alors cette disquette dans le lecteur A:, et on tapera A:INSTALL pour lancer la procédure d'installation.

Le sous-répertoire ISP contient le logiciel offert par Lattice pour la programmation des ispGAL22V10 et des ispLSI (© Lattice Semiconductor Corp.) Il peut être directement installé à partir de la disquette du livre en exécutant, depuis Windows, le fichier SETUP.EXE du répertoire ISP.

Le sous-répertoire PIC, enfin, contient le logiciel de pilotage des programmeurs de PIC16C84 et d'EEPROM série décrits au chapitre 5, que nous avons téléchargé pour vous sur Internet (© 1995, Silicon Studio).

Il suffit de le recopier dans un répertoire du disque dur ou sur une disquette de travail pour pouvoir l'utiliser immédiatement, sans procédure d'installation préalable, du moins si le programmeur est raccordé au port série COM1: du PC (taper alors PIP).